UNIVERSITÉ DE MONTRÉAL

TECHNIQUES IN ORDINAL CLASSIFICATION AND IMAGE-TO-IMAGE TRANSLATION

CHRISTOPHER BECKHAM
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2017

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

TECHNIQUES IN ORDINAL CLASSIFICATION AND IMAGE-TO-IMAGE TRANSLATION

présenté par: BECKHAM Christopher
en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de:

Mme CHERIET Farida, Ph. D., présidente
M. PAL Christopher J., Ph. D., membre et directeur de recherche
M. CHARLIN Laurent, Ph. D., membre

ACKNOWLEDGEMENTS

I am very grateful and privileged to be part of the development of a new frontier in machine learning research, one which gives countless opportunities to transform the world we live in and make it a better place for everyone to live in. Undertaking this masters degree has allowed me to build a solid foundation which will benefit me in the years to come. I would like to thank my advisor, Christopher Pal, for having recruited me on his sabbatical in New Zealand, even if it did mean sacrificing warm beaches for sub-zero temperatures and winter blues.

I would also like to thank Samsung and Imagia Cybernetics for funding (respectively) the first and second years of this degree.

RÉSUMÉ

Dans cette thèse, nous explorons deux thèmes de recherche dans le domaine de l'apprentissage en profondeur et de l'imagerie médicale. La première est dans la classification ordinale, dans laquelle les classes à prévoir sont discrètes mais ont une relation d'ordonnancement. Les distributions de probabilités sous les classes ordinales peuvent posséder des propriétés indésirables, comme la non-unimodalité. Nous proposons une technique simple pour contraindre les distributions de probabilités ordinales discrètes à être unimodales par l'utilisation des distributions de Poisson et des distributions de probabilités binomiales. Nous évaluons cette approche sur la base d'une estimation de l'âge et d'un ensemble de données Kaggle sur la rétinopathie diabétique et obtenons des résultats compétitifs. Nous supposons que la contrainte d'unimodalité – en plus de rendre les distributions de probabilité plus interprétables – agit comme un régularisateur qui peut atténuer le dépassement, surtout dans un régime de données faible. Dans le second thème, nous explorons la traduction d'image à image contradictoire et motivons leur utilité dans le cadre d'un apprentissage semi-supervisé. Nous évaluons une méthode existante et en proposons une nouvelle que nous évaluons sur plusieurs bases de données comme celles utilisées dans notre travail sur la classification ordinale. Dans ce dernier cas, nous voulons établir une correspondance entre le domaine des scanners de patients symptomatiques et celui des scanners de patients non symptomatiques. Cela forme effectivement un modèle qui peut démêler les facteurs de variation sous-jacents et apprendre à détecter et à supprimer les zones symptomatiques de l'image, ce qui pourrait être exploité de plusieurs façons, comme aider un réseau qui s'appuie sur des étiquettes riches, ou générer des exemples synthétiques. Nous présentons des résultats qualitatifs intéressants et motivons plusieurs pistes prometteuses pour l'avenir.

ABSTRACT

In this thesis we explore two research topics within the realm of deep learning and medical imaging. The first is in ordinal classification, in which the classes to be predicted are discrete but have an ordering relation. Probability distributions under ordinal classes can possess undesired properties, such as non-unimodality. We propose a straightforward technique to constrain discrete ordinal probability distributions to be unimodal via the use of the Poisson and binomial probability distributions. We evaluate this approach on an age estimation and Kaggle diabetic retinopathy dataset and obtain competitive results. We conjecture that the unimodality constraint – in addition to making the probability distributions more interpretable – acts as a regulariser which can mitigate overfitting, especially in a low data regime. In the second topic, we explore adversarial image-to-image translation and motivate their utility within the framework of semi-supervised learning. We evaluate an existing method and propose a new one which we evaluate on several datasets such as the ones employed in our work on ordinal classification. In the case of the latter, we want to map from the domain of symptomatic patient scans to non-symptomatic patient scans. This effectively trains a model which can disentangle the underlying factors of variation and learn to detect and remove symptomatic regions in the image, which could be leveraged in several ways, such as aiding a network which relies on rich labels, or generating synthetic examples. We present some interesting qualitative results and motivate several promising avenues to take for the future.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

MNIST	Benchmark dataset for digit classification (acronym stands for Mixed National Institute of Standards and Technology
MLP	Multi-layer perceptron
CNN	Convolutional neural network
PMF	Probability mass function
EMD	Earth mover's distance
POM	Proportional odds model
POMNN	Proportional odds model neural network
QWK	Quadratic weighted kappa
GAN	Generative adversarial network

LISTE DES ANNEXES

CHAPTER 1 INTRODUCTION

Deep learning has emerged as one of the most exciting fields of study one could pursue research in within the realm of machine learning. The foundations which uphold deep learning – *neural networks* and their associated algorithms to train them – have been around for quite some time, dating back as far as the eighties. Neural networks are a class of machine learning algorithms which – in a very loose manner of speaking – try to imitate some of the processes which occur in our brains when we learn and process information. At the heart of deep learning is a notion called 'representation learning', which, concretely, describes algorithms which are fed raw data and try to learn useful features which can be leveraged to perform some sort of task such as classification. These features can be hierarchical in nature, in the sense that highly abstract features can be learned based on more lowly abstract features which in turn are based on the raw data. In the context of neural networks, this is possible by constructing multiple layers where the computation at each layer in the network is some non-linear transformation of its input[1]. For example, in the case of object recognition, the raw data would simply comprise a matrix of numbers denoting the raw pixel intensities in the image, but the next layer may learn simple detectors which can then be combined in the following layer to construct the outlines of objects of interest. Given enough data these layers become effective at disentangling what is important in the image from what is not, i.e. becoming robust to noise. Indeed, we as humans do not perceive the world as a matrix of numbers, but rather as very abstract entities comprised of lower-abstract entities which is what we try to achieve with representation learning.

In the conventional machine learning framework we do have representation learning, but rather some sort of algorithm which takes features of input and produces a classification. The question then becomes, where do the features come from in the first place? For extremely simple tasks, one could simply use the raw data as features and achieve decent results. For more complicated tasks such as image classification, using the raw data – which in this case, are the raw pixel intensities – simply would not work. To address this, a lot of time is spent 'hand-engineering' features which we believe would be useful to make a classification. For example, in the case of spam classification (i.e., is this email spam or not?), we may need to pre-process our corpus of emails and try to extract features which we think are useful, such as the frequency of the most commonly occuring words, whether the subject is in all-caps, whether the word 'pharmacy' is mentioned, and so forth. While this effort may produce

[1]This distinction is key, since multiple linear transformations could simply be collapsed into one layer!

an effective spam classifier, it is certainly not scalable; to effectively come up with useful features requires significant human effort in engineering these features, and as the dataset increases in size it becomes harder to engineer features with low generalisation error (not all spam emails are about pharmaceuticals, and some legitimate emails are!). In the framework of deep neural networks (which are an instance of representation learning), we would instead have an algorithm which could be fed the raw corpus of text – as well as their labels denoting which text is spam or not – and the algorithm would be able to operate on this directly, learning multiple hierarchies of features which could be leveraged to make a classification. It could initially learn very low-level features such as whether certain words are present or not (like 'pharmacy') to more complicated features such as whether certain motifs are present in conjunction with other motifs in the sentence (i.e. high-level features). If we examine the internal workings of the network, we could expect that the network would indeed learn very interesting combinations of features, ones which we would never have thought to come up with!

Just like conventional machine learning, we are able to build the best models when we have a rich collection of labels available, which is the case of supervised learning. In this framework we have a label available with every data point, and we train our model to produce some sort of decision boundary to classify whether a data point belongs in one class or not, using the labels to determine how effective that decision boundary is. Of course, in practice, it is not always possible to obtain labels because certain labels are more expensive to obtain than others. For example, labeling a dataset of cats versus dogs is a much easier process than having a radiologist segment (i.e., produce a pixel-wise annotation) cancerous regions in a tissue scan. Because of this, significant efforts have been made in *unsupervised learning*, which is where we try to learn interesting features from the data without any labels whatsoever. In reality, whether a task is supervised vs unsupervised is really a spectrum, and this is for two reasons: firstly, one could mix both labeled and unlabeled data to produce a model (which s called *semi-supervised learning*); and secondly, some labels are richer than others. For example, a segmentation is a pixel-wise annotation which contains significantly more information than a bounding box (a square annotating the region of interest) which is more expensive than a per-image label. Therefore, even the term 'supervised' can reflect a spectrum in of itself, where the data could be very strongly supervised to very weakly supervised. In fact, we discuss both unsupervised learning and (very weakly) supervised learning in Chapter 4.

In part, what has enabled resurgence and popularity of neural networks has been a combination of big data, theory and hardware which can perform computations on a massively parallel scale. For example, graphics processing units (GPUs), which used to only be used

primarily for rendering high-resolution movies and video games to the screen, has been leveraged to perform certain neural network computations which can be sped-up immensely from being parallelised (for example, matrix-matrix multiplication and convolutions). In terms of theoretical developments, simple tweaks to how neural networks were trained (which in retrospect were quite simple tweaks!) allowed researchers to train deeper networks with fewer stability issues, and since they're deeper we end up learning more abstract features. These breakthroughs – in addition to a wealth of data available to us today – has allowed us to significantly beat benchmarks in many real-world applications, such as object detection and localisation, natural language processing and translation, video annotation, autonomous agents, and medical imaging. Deep learning has made a significant impact on almost every field of research, and there are enormous investments from major companies which all see the utility in incorporating artificial intelligence into their products.

In this thesis we will be exploring two research topics in deep learning with strong applicability to medical imaging and diagnosis. In Chapter 3 we discuss ordinal classification, in which the classes we wish to classify a problem as contain some sort of ordering; this is quite important in practice because ordered data implies that certain misclassifications are worse than others. Because of this, we want to ensure that we focus on mitigating risky classifications which could in practice result in costly clinical diagnoses. In particular, our contribution is in imposing a unimodal constraint on the probability distributions, which not only makes them interpretable in an ordinal context but also imposes a regularising effect on the distribution itself, which is especially important to consider in a low data regime. In Chapter 4 we explore generative adversarial networks and their application in image-to-image translation, and propose some interesting avenues of research where one could learn more powerful models by incorporating unlabeled data (i.e., semi-supervised learning), which in practice is quite abundant and cheaper to obtain in comparison to labeled data. Unlike Chapter 3 however, we do not present any published work of our own, but rather a formidable exploration into image-to-image translation and promising avenues of exploration which we would like to explore in the months to come.

Both of these topics we present shed light on a common issue in deep learning, which is data scarcity, which can be in terms of the data itself, or the supervisory signals associated with them (the labels). For example, in Chapter 3 we propose a technique to constrain probability distributions to be unimodal; since this effectively limits the expressivity of the network, it can be seen as a form of regularisation, the definition of which is any technique which can be used to reduce overfitting in neural networks. In Chapter 4, we motivate the use of semi-supervised learning, in which we try to incorporate unlabeled data (or weakly labeled data) in order to help build more expressive and powerful models. This field of deep learning has

attracted enormous interest due to the fact that labeled data (or richly labeled data) can be expensive to obtain, depending on the problem domain.

In the next chapter we introduce various concepts which will comprise a formidable foundation on which one can understand deep learning and recognise its connection to its more traditional counterparts in classical machine learning.

CHAPTER 2 CRITICAL LITERATURE REVIEW

2.1 Computational graphs

This section has been inspired by Christopher Olah's blog post 'Calculus on Computational Graphs', which can be found here[1].

We start off by introducing the notions behind computational graphs, why they are important in deep learning, and how we can use them to perform inference on neural networks. Following this section we will introduce some of the most commonly used deep neural networks (such as multilayer perceptrons, convolutional neural networks, and autoencoders) and other concepts which will be useful to know for the remainder of this thesis. While the nitty-gritty details behind computational graphs have fortunately been abstracted away in modern frameworks, it is hoped that taking this approach will at least give the reader a sense of appreciation for what is happening behind the scenes when one is performing inference on neural network models.

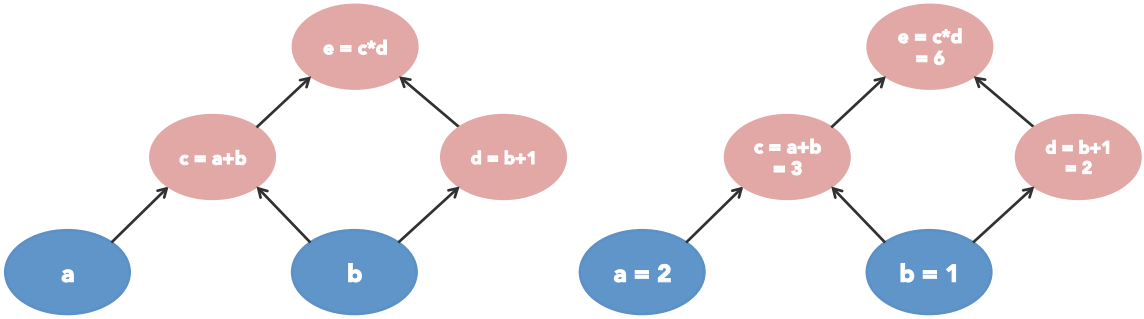
When we write computer programs we are used to writing code in an imperative fashion, where we may do things such as define input variables, intermediate variables, or changing the state of other variables. For the same of simplicity let us adopt a more functional paradigm and not concern ourselves with being able to modify existing state. Let us define a really simple function, which takes some arguments (our inputs) and computes some intermediate variables in terms of summations and multiplications.

```
def some_function(a,b): # inputs are 'a' and 'b'
    c = a+b
    d = b+1
    e = c*d
    return e
```

We can also express this computation in terms of an abstract syntax tree. We present a very simplified one below, in which the nodes represent either variables or intermediate computations.

Why is this representation useful? When we lay out our computation in this tree structure, we can easily compute derivatives of variables (with respect to any other variables) through a simple bottom-up or top-down traversal, which is useful because we would like to be able to

[1] http://colah.github.io/posts/2015-08-Backprop/



(a) Symbolic definition (b) Execution of the graph

Figure 2.1 Computational graph generated from the `c = a+b; d = b+1; e = c*d`. Blue nodes are inputs to the graph, and red nodes are intermediate variables/computations.

quantify the effect that the perturbation of a particular variable has on subsequent variables in the graph. In doing so, there are only three simple rules from calculus we need to remember: the sum rule, product rule, and chain rule:

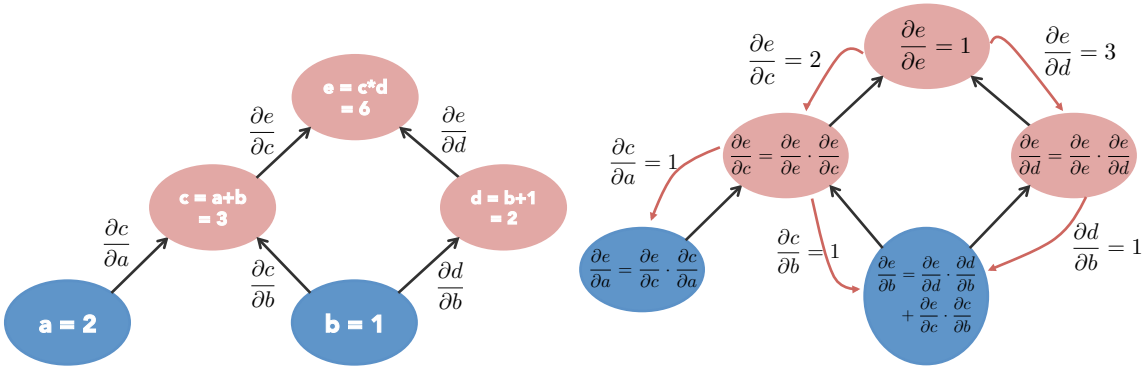
$$\begin{aligned} \frac{\partial(a+b)}{\partial a} &= \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a} \text{ (sum rule)} \\ \frac{\partial uv}{u} &= u \cdot \frac{\partial v}{\partial u} + v \cdot \frac{\partial u}{\partial u} \text{ (product rule)} \\ \frac{\partial\ f(g(x))}{\partial x} &= \frac{\partial\ f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x} \text{ (chain rule)} \end{aligned} \tag{2.1}$$

Once we have initialised our initial variables a and b and computed the intermediate variables (as seen in Figure 2.1(b)), we can compute the derivative of the output of every node in the graph with respect to its input(s), by starting off from a and b and working from the bottom-up. The result of this can be seen in Figure 2.2(a), where the derivatives are shown on the edges connecting nodes in the graph; for example on the edge from $a \to c$ we have $\frac{\partial c}{\partial a}$, from $c \to e$ we have $\frac{\partial e}{\partial c}$ and so forth.

Let's say we wanted to compute $\frac{\partial e}{\partial b}$, i.e., the derivative of an intermediate variable with respect to one of the two inputs, b. We already have all the necessary derivatives we need. We just compute it using the chain rule:

$$\frac{\partial e}{\partial b} = \underbrace{\frac{\partial e}{\partial d}}_{\text{from d to e}} \cdot \underbrace{\frac{\partial d}{\partial b}}_{\text{from b to d}} = 1 \cdot 3 = 3$$

If we were to apply the chain rule in this fashion, computing the derivative of every node with



(a) Derivatives obtained from forward-mode differentiation (i.e., we want the derivative of every intermediate variable w.r.t. to all inputs)

(b) Derivatives obtained from reverse-mode differentiation (i.e., we want the derivative of every output w.r.t every input)

Figure 2.2 Computational graph generated from the `c = a+b; d = b+1; e = c*d` and the corresponding derivatives computed from forward mode (left) and reverse-mode (backpropagation) approach (right).

respect to a and the derivative of every node with respect to b as we traverse the graph, we would have the derivative of every expression with respect to all of the inputs. This is called *forward-mode differentiation*. There is also another form, called *reverse-mode differentiation*. In this mode, we do not want the derivative of every expression with respect to the input – we want the derivative of every output with respect to every expression! The result of this is shown in Figure 2.2(b) (which can be contrasted with the forward-mode presented in Figure 2.2(a)). For concreteness, we will do an example where we want to obtain the derivative of the output e to respect to one of the inputs, b.

When we start from the final output e, we only have one derivative, which is simply $\frac{\partial e}{\partial e} = 1$. Suppose we traverse down from $e \to d$. From this, we can compute the derivative $\frac{\partial e}{\partial d} = \frac{\partial e}{\partial e}\frac{\partial e}{\partial d}$. We want to use this derivative to help us compute $\frac{\partial e}{\partial b}$, which is the derivative of the output with respect to our input of interest b. Through the chain rule, we know that the only thing left we need to compute that is to also compute $\frac{\partial d}{\partial b}$, such that:

$$\underbrace{\frac{\partial e}{\partial b}}_{\text{what we want}} = \underbrace{\frac{\partial e}{\partial d}}_{\text{we already have}} \cdot \underbrace{\frac{\partial d}{\partial b}}_{\text{we need to compute}},$$

where $\frac{\partial d}{\partial b}$ is computed by making a traversal from $d \to b$. Suppose we do another example and start off by traversing from $e \to c$. At this node we compute $\frac{\partial e}{\partial c} = \frac{\partial e}{\partial e}\frac{\partial e}{\partial c}$, but we need to

compute $\frac{\partial e}{\partial b}$. So what we need is $\frac{\partial c}{\partial b}$, such that:

$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial b},$$

where $\frac{\partial c}{\partial b}$ can be computed by going from $c \to b$.

But now, we notice something interesting: we actually have two $\frac{\partial e}{\partial b}$ derivatives! That is ok, and it makes sense, since b affects the final outcome e from two different pathways ($b \to c$ and $b \to d$). In this case, we just sum the two derivatives together and this is our final $\frac{\partial e}{\partial b}$.

We can see that whenever we traverse down from one node to another, say, from $y \to F(x)$ (where $y = F(x)$) there are two things we always have to end up computing. The first is the derivative of the output of that node with respect to its input, which is simply $\frac{\partial F(x)}{\partial x}$. The second is computing the derivative of some output node L at the top of the graph with respect to that input x, which we can simply do since we also got passed the gradient $\frac{\partial L}{\partial F(x)}$. That means we can simply compute $\frac{\partial L}{\partial x} = \frac{\partial F(x)}{\partial x}\frac{\partial L}{\partial F(x)}$ and repeat the exact same process as we descend down the graph.

In the next section we will explore linear regression, one of the simplest machine learning algorithms. To build on our simple example in this section we will also view it and optimise it in terms of a computational graph.

2.2 Linear regression

Let's do the simplest machine learning algorithm that we know: linear regression. We assume that x and y are scalars and therefore we wish to learn some parameters w and b such that we minimise the mean squared error over training data $\{x_i, y_i\}_{i=1}^n$, where $x_i \in \mathbb{R}$ and $y_i \in \mathbb{R}$:

$$\min_\theta L(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(x_i, y_i) = \frac{1}{n}\sum_{i=1}^{n} (f(x_i) - y_i)^2, \tag{2.2}$$

where $f(x) = wx + b$ and $\theta = \{w, b\}$. We can update the parameters w and b if we know their derivatives $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial b}$ by performing gradient descent. For example, in the case of stochastic gradient descent, we process each example $\{x_i, y_i\}$ one-by-one to update the weights θ:

$$w := w - \alpha \frac{\partial \ell(x_i, y_i)}{\partial w},\ b := b - \alpha \frac{\partial \ell(x_i, y_i)}{\partial b} \tag{2.3}$$

While the loss in Equation 2.2 is a mean over the examples in the training set, we present a simple computational graph in which we are simply minimising the loss over one training

example $\ell(x, y) = (f(x) - y)^2$, which can be seen in Figure 2.3. In that case, to actually compute the gradient in Equation 2.2 we simply compute the gradients individually for every example and average them.

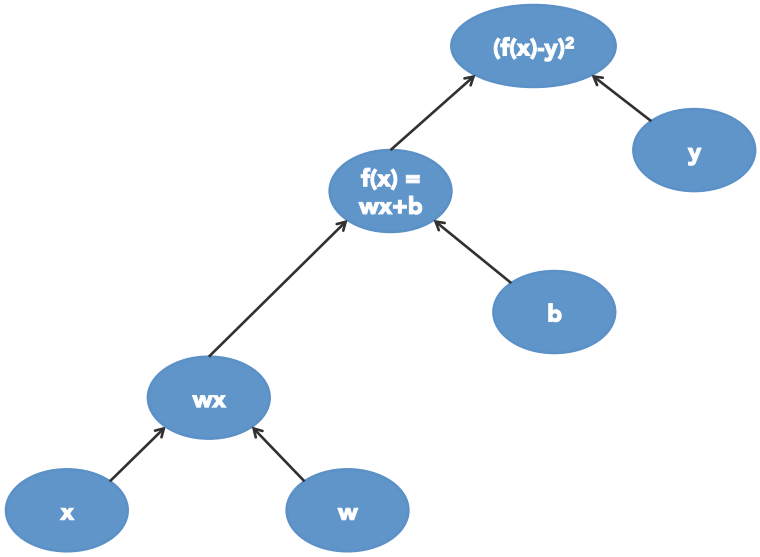


Figure 2.3 Linear regression expressed as a computational graph. x, w, and b are inputs to the graph.

Because we want to obtain $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial b}$, we do not need to go from $(f(x)-y)^2 \rightarrow y$ in the graph. That means we go from $(f(x) - y)^2$ to $f(x)$, which means we need to compute $\frac{\partial\ (f(x)-y)^2}{\partial\ f(x)}$. We can do this via the chain rule:

$$\frac{\partial\ \ell}{\partial\ f(x)} = \frac{\partial\ (f(x) - y)^2}{\partial\ f(x)} = \frac{\partial\ (f(x) - y)^2}{\partial\ (f(x) - y)} \cdot \frac{\partial\ (f(x) - y)}{\partial\ f(x)} = 2(f(x) - y) \cdot 1$$

From $f(x)$, we can then traverse down to the b node and compute $\frac{\partial f(x)}{\partial b}$, which is what we want to do since we need to compute $\frac{\partial\ \ell}{\partial\ b}$. Again, through the chain rule we can compute this easily:

$$\begin{aligned} \underbrace{\frac{\partial\ \ell}{\partial\ b}}_{\text{what we want}} &= \underbrace{\frac{\partial\ \ell}{\partial\ f(x)}}_{\text{what we have}} \cdot \underbrace{\frac{\partial\ f(x)}{\partial b}}_{\text{need to compute}} \\ &= 2(f(x) - y) \cdot 1 \\ &= 2(f(x) - y) \end{aligned} \tag{2.4}$$

We now have an update expression for the bias! We will come back to this later. For now, we need to also compute the same thing for the weight w.

Now we are only left with finding $\frac{\partial \ell}{\partial w}$. Traversing backwards from $f(x) \to wx$, we need to compute $\frac{\partial \ell}{\partial wx}$:

$$\begin{aligned} \frac{\partial\ \ell}{\partial\ wx} &= \frac{\partial\ \ell}{\partial\ f(x)} \cdot \frac{\partial\ f(x)}{\partial wx} \\ &= 2(f(x) - y) \cdot 1 \end{aligned} \tag{2.5}$$

Then from $wx \to w$ another application of chain rule:

$$\begin{aligned} \frac{\partial\ \ell}{\partial\ w} &= \frac{\partial\ \ell}{\partial\ wx} \cdot \frac{\partial\ wx}{\partial w} \\ &= 2(f(x) - y) \cdot x \end{aligned} \tag{2.6}$$

We are now finished! From this we have obtained our desired derivatives:

$$\begin{aligned} \frac{\partial\ \ell}{\partial\ b} &= 2(f(x) - y) \\ \frac{\partial\ \ell}{\partial\ w} &= 2(f(x) - y) \cdot x \end{aligned} \tag{2.7}$$

To find parameters w and b to minimise the loss $(f(x) - y)^2$, we can then utilise gradient descent as we had done in Equation 2.3:

2.3 Multivariate linear regression

Now we generalise linear regression to the multivariate case, where we now have inputs $(\mathbf{x}_i, y_i)_{i=1}^n$ where $\mathbf{x} \in \mathbb{R}^p$ (for p variables) and $y \in \mathbb{R}$. Because $\mathbf{x}$ is now a vector instead of a scalar, this means we now have a weight vector $\boldsymbol{\theta}$ in addition to scalar bias b such that $f(\mathbf{x}) = \theta^T \mathbf{x} + b$. To keep things simple however, for this example we will omit the bias b. Because $\mathbf{x}$ and $\boldsymbol{\theta}$ are now vectors, we will also have to deal with derivatives which are also vectors. Again, to keep things simple, we will present the backpropagation based on a single example $\{\mathbf{x}, y\}$.

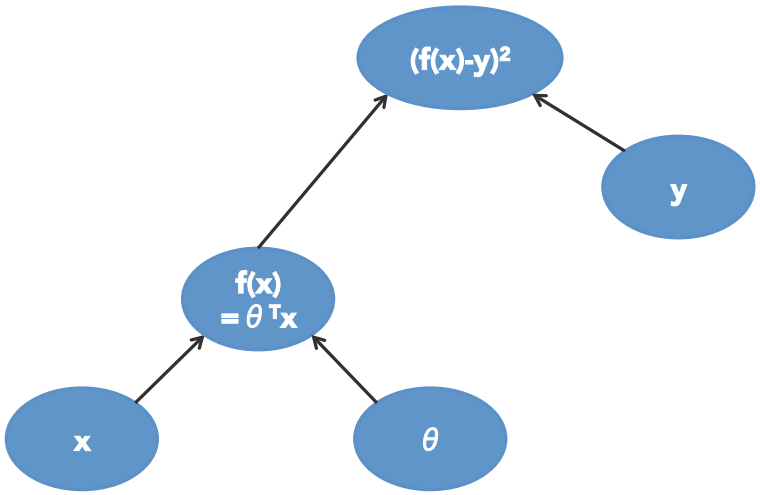


Figure 2.4 Computational graph of multivariate linear regression $f(\mathbf{x}) = \boldsymbol{\theta}^T\mathbf{x}$ (bias omitted for brevity)

Again, let's compute the derivatives that we need to update $\boldsymbol{\theta}$. Traversing backward from $(f(\mathbf{x}) - y)^2 \rightarrow f(\mathbf{x})$ we want to compute $\frac{\partial(f(\mathbf{x})-y)^2)}{\partial f(\mathbf{x})}$:

$$\frac{\partial\ (f(\mathbf{x}) - y)^2}{\partial\ f(\mathbf{x})} = \frac{\partial\ (f(\mathbf{x}) - y)^2}{\partial\ (f(\mathbf{x}) - y)} \odot \frac{\partial\ (f(\mathbf{x}) - y)}{\partial\ f(\mathbf{x})} = 2(f(\mathbf{x}) - y) \odot 1$$

We note that this is actually a vector of derivatives of length p (corresponding to p features), rather than just a scalar! That is:

$$\frac{\partial\ (f(\mathbf{x}) - y)^2}{\partial\ f(\mathbf{x})} = \Big[2(f(\mathbf{x}_1) - y), \ldots, 2(f(\mathbf{x}_p) - y)\Big]$$

Continuing forward, from $f(\mathbf{x}) \rightarrow \boldsymbol{\theta}$ we wish to compute:

$$\frac{\partial\ \boldsymbol{\theta}^T\mathbf{x}}{\partial\ \boldsymbol{\theta}} = \Big[\frac{\partial\ \boldsymbol{\theta}^T\mathbf{x}}{\partial\ \boldsymbol{\theta}_1}, \ldots, \frac{\partial\ \boldsymbol{\theta}^T\mathbf{x}}{\partial\ \boldsymbol{\theta}_p}\Big]$$

This derivative is also a scalar! While $\boldsymbol{\theta}^T\mathbf{x}$ is a scalar (since it is just a dot product), $\boldsymbol{\theta}$ is a weight vector that is p units in length, so the resulting derivative is also p units in length. For concreteness we expand this vector to show all the elements like so: To simplify this vector of derivatives, we need to know what the derivative is for the i'th element, i.e., $\frac{\partial\ \boldsymbol{\theta}^T\mathbf{x}}{\partial\ \boldsymbol{\theta}_i}$. So let us consider one of these cases. Suppose we are interested in $\boldsymbol{\theta}_i$: we expand out the following dot product:

$$\frac{\partial\ \boldsymbol{\theta}^T\mathbf{x}}{\partial\ \boldsymbol{\theta}_i} = \frac{\partial\ \boldsymbol{\theta}_1\mathbf{x}_1 + \cdots + \boldsymbol{\theta}_i\mathbf{x}_i + \cdots + \boldsymbol{\theta}_p\mathbf{x}_p}{\partial\ \boldsymbol{\theta}_i}$$

We notice that in the numerator, only the $\boldsymbol{\theta}_i \mathbf{x}_i$ term matters, since $\frac{\partial \boldsymbol{\theta}_j \mathbf{x} j}{\partial \boldsymbol{\theta}_i} = 0$ for $i \neq j$. Therefore, we can simplify the expression to:

$$\frac{\partial\ \boldsymbol{\theta}^T \mathbf{x}}{\partial\ \boldsymbol{\theta}_i} = \frac{\partial\ \boldsymbol{\theta}_1 \mathbf{x}_1 + \cdots + \boldsymbol{\theta}_i \mathbf{x}_i + \cdots + \boldsymbol{\theta}_p \mathbf{x}_p}{d\ \boldsymbol{\theta}_i} = \frac{\partial\ \boldsymbol{\theta}_i \mathbf{x}_i}{\partial\ \boldsymbol{\theta}_i} = \mathbf{x}_i$$

This means that we can write $\frac{\partial\ \boldsymbol{\theta}^T \mathbf{x}}{\partial\ \boldsymbol{\theta}}$ in vector form very easily!

$$\frac{\partial\ \boldsymbol{\theta}^T \mathbf{x}}{\partial\ \boldsymbol{\theta}} = \Big[\mathbf{x}_1, \ldots, \mathbf{x}_p\Big] = \mathbf{x}$$

We can now write the full expression to compute the gradient of the loss w.r.t. θ.

$$\frac{\partial\ (f(\mathbf{x}) - y)^2}{\partial\ \boldsymbol{\theta}} = \frac{\partial\ (f(\mathbf{x}) - y)^2}{\partial\ f(\mathbf{x})} \odot \frac{\partial\ f(\mathbf{x})}{\partial \boldsymbol{\theta}} = 2(f(\mathbf{x}) - y) \odot \mathbf{x}$$

Note that these derivatives get slightly more complicated if we have to deal with matrices of derivatives, which for example would have happened if instead $\mathbf{x}$ was an $n \times p$ input matrix) consisting of multiple examples. For instance, if $\mathbf{x}$ comprised n examples (i.e. a minibatch) then $\theta^T \mathbf{x}$ would be an $(n \times 1)$ matrix, and if we were to compute $\frac{\partial \theta^T \mathbf{x}}{\partial \theta}$ it would end up being an $(n \times p)$ matrix, as shown below:

$$\frac{\partial \boldsymbol{\theta}^T \mathbf{x}}{\partial \boldsymbol{\theta}} = \begin{bmatrix} \frac{\partial \boldsymbol{\theta}^T \mathbf{x}_1}{\partial \boldsymbol{\theta}} \\ \vdots \\ \frac{\partial \boldsymbol{\theta}^T \mathbf{x}_n}{\partial \boldsymbol{\theta}} \end{bmatrix} = \begin{bmatrix} \frac{\partial \boldsymbol{\theta}^T \mathbf{x}_1}{\partial \boldsymbol{\theta}_1}, \ldots, \frac{\partial \boldsymbol{\theta}^T \mathbf{x}_1}{\partial \boldsymbol{\theta}_p} \\ \vdots \\ \frac{\partial \boldsymbol{\theta}^T \mathbf{x}_n}{\partial \boldsymbol{\theta}_1}, \ldots, \frac{\partial \boldsymbol{\theta}^T \mathbf{x}_n}{\partial \boldsymbol{\theta}_p} \end{bmatrix} \tag{2.8}$$

In these cases, one has to carefully ensure that the matrices being multiplied are in the correct order (since matrix multiplication is not commutative) and that the resulting dot product (whether that be a matrix-vector or matrix-matrix dot product) makes sense semantically. However, for the sake of brevity we will leave away further details regarding the backpropagation in following sections.

2.4 Logistic regression

We now present logistic regression, which is a slight modification of linear regression in which we wish to predict a discrete label (i.e., classification) rather than a continuous value (regression). For now, let $y \in \{0, 1\}$ be a binary value as we will start off with binary classification and then extend it to multi-way..

To be succinct with notation, we define our input $\mathbf{x} \in \mathbb{R}^p$, label $y \in \{0, 1\}$, and weight vector

and bias $\boldsymbol{\theta} \in \mathbb{R}^p, b \in \mathbb{R}$, such that we can define the probability of the positive class:

$$p(y = 1 \mid \mathbf{x}) = sigm(\boldsymbol{\theta}^T \mathbf{x}),$$

where $sigm(\mathbf{z}) = \frac{1}{1+\exp(-\mathbf{z})}$. This function is called either a *nonlinearity* or an *activation function*. In our case, it is useful as this particular function is a mapping $\mathbb{R} \to \{0, 1\}$ which is precisely what we need in order to represent a probability.[2] For the sake of succinctness, we will allow $p(y|\mathbf{x})$ to refer to $p(y = 1|\mathbf{x})$.

Instead of using squared error as the loss function as we did in the linear regression case, we instead maximise a loss function called the cross-entropy:

$$\ell(\mathbf{x}, y) = y \cdot \log(p(y|\mathbf{x})) + (1 - y) \cdot \log(1 - p(y|\mathbf{x})) \tag{2.9}$$

While we omitted the derivation behind this loss function, the result is hopefully intuitive upon close inspection: when $y = 1$, the right-most term cancels out and we are left with the left-hand term $\log(p(y|\mathbf{x}))$. The log is simply there for the purpose of numeric stability and since the log function is monotonic, maximising the log of a term is equivalent to maximising the term itself. Similarly, we can see that if $y = 0$ the right-hand term stays and the other disappears, and in this we try to maximise (the log of) $1 - p(y|\mathbf{x})$, which is the probability of y being zero.

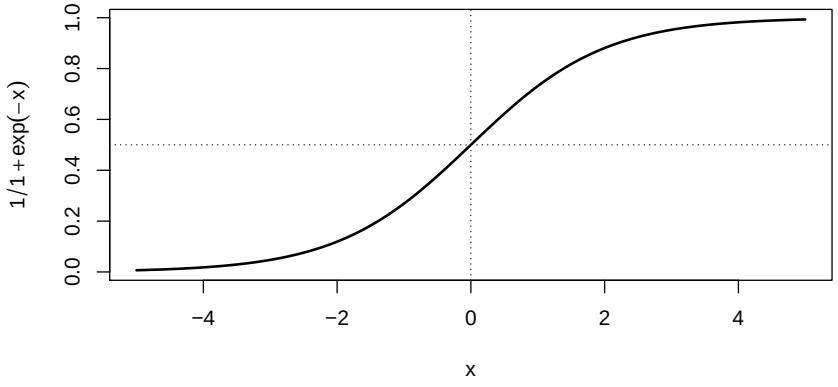
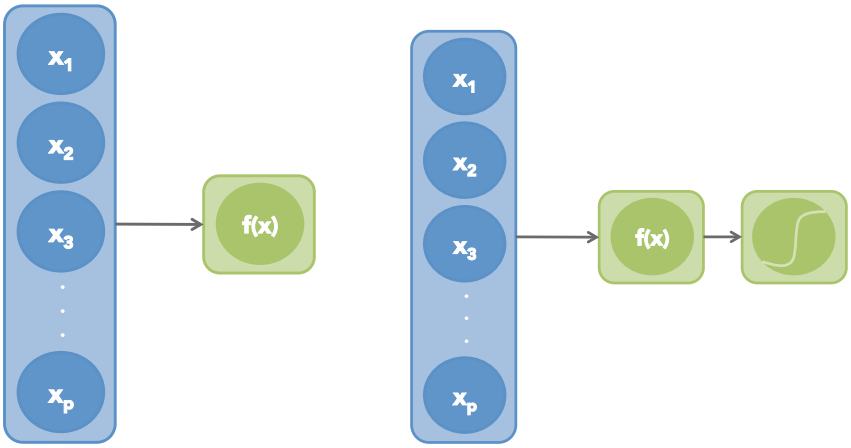


Figure 2.5 Plot of the sigmoid nonlinearity: $\frac{1}{1+exp(-x)}$

[2]Other nonlinearities exist, which have different properties. For example, tanh $= \frac{exp(x)-exp(-x)}{exp(x)+exp(-x)}$ normalises its inputs to be in the range $[-1, +1]$. The rectified linear unit, relu$(x) = \max(0, x)$ only retains positive values and zeros out negative ones.

In Figure 2.6 we visualise linear and logistic regression in terms of 'layers', which will be useful for when we introduce multilayer perceptrons in the next section.



(a) Linear regression (b) Logistic regression

Figure 2.6 Layer-wise representation of linear regression (left) and logistic regression (right). We can think of the inputs $\mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_p\}$ comprising an input layer (blue), which is fully-connected to a final layer $f(\mathbf{x})$ consisting of one unit. For logistic regression, this prediction $f(\mathbf{x})$ is followed by an element-wise sigmoid nonlinearity.

2.4.1 Multinomial logistic regression

We now extend the above to account for multiple classes, i.e., $k \in \{0, \ldots, K-1\}$, where K denotes the number of classes. This means that instead of having one output determining $p(y = 1|\mathbf{x}$ we instead have K outputs. This means we also have to change the label y to a vector $\mathbf{y} \in \{0, 1\}^K$. For some class k this vector will be all zeros except for the k'th element; this type of vector is called a 'one-hot' vector, since it is only 'activated' at one position. In order to fall more in line with common notation used in deep learning, we now substitute the earlier $\boldsymbol{\theta}$ for $\mathbf{W}$, which is now a matrix. In fact, instead of being a vector of p weights, it is now a $p \times K$ matrix of weights, where $\mathbf{W}_{\cdot i}$ – the i'th column – denotes weights corresponding to $p(y = i|\mathbf{x})$. The bias also changes from a scalar b to $\mathbf{b} \in \mathbb{R}^K$, where there is now a bias for every class probability. We can define the probability of the k'th class as:

$$p(y = k|\mathbf{x}) = \text{softmax}(\mathbf{x}\mathbf{W} + \mathbf{b}) \in [0, 1]^K, \tag{2.10}$$

where softmax($\mathbf{z}$) is an activation function which ensures the sum of its inputs is 1 by performing the following operation:

$$\text{softmax}(\mathbf{h})_i = \frac{\exp(\mathbf{h}_i)}{\sum_{j=1}^{K} \exp(\mathbf{h}_j)} \tag{2.11}$$

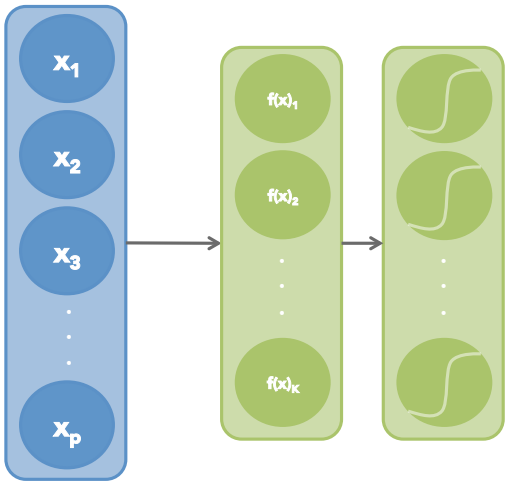


Figure 2.7 Layer representation of multinomial logistic regression. Instead of an output layer with only one unit (as shown in Figure 2.6(b)) we now have multiple outputs $f(\mathbf{x}) = \{f(\mathbf{x})_1, \ldots, f(\mathbf{x})_K\}$ for K classes. This layer is then fed into the softmax function to produce $p(\mathbf{y}_i|\mathbf{x})$ (and enforce the constraint that these probabilities sum to one). Note that for the sake of concreteness, we have separated out the linear transformation (first green layer) from the nonlinearity function (second green layer). It is common however to merge both of these into one layer.

The corresponding loss function is also cross-entropy just like in Equation 2.9 but generalised to K classes instead:

$$\ell(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{K} \mathbf{y}_i \cdot \log(p(\mathbf{y}_i|\mathbf{x})) \tag{2.12}$$

Note that since $\mathbf{y}$ is a one-hot vector, whose only non-zero element for the k'th class is at the k'th element, the above loss can simply be simplified to $\log(p(\mathbf{y}_c|\mathbf{x}))$, where c denotes the class of y.

2.5 Multilayer perceptrons

We can take this formulation insert more layers in between. Let $\mathbf{h}^{(1)} \in \mathbb{R}^m$ denote the first hidden layer consisting of m units. Since we may have multiple hidden layers $l \in \{1, \ldots, L-1\}$ (where L denotes the final layer) we will also define the weight and bias parameters of the l'th layer as $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$, respectively. We can then define the l'th hidden layer as such:

$$\mathbf{h}^{(l)} = g(\mathbf{h}^{(l-1)}\mathbf{W}^{(l)} + \mathbf{b}^{(l)}),$$

where $\mathbf{W}^{(l)} \in \mathbb{R}^{(n_{l-1} \times n_l)}$, $\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$ and n_l denotes the number of units in a layer (n_0 corresponds to the number of input features in $\mathbf{x}$, and likewise $\mathbf{h}^{(0)} = \mathbf{x}$). Then, we define the final layer $h^{(L)} \in \mathbb{R}^K$ as (in the case of classification):

$$\mathbf{h}^{(L)} = \text{softmax}(\mathbf{h}^{(L-1)}\mathbf{W}^{(L)} + \mathbf{b}^{(L)})$$

This notion allows us to motivate one of the reasons why deep neural networks are quite effective at what they do. While the precise definition is still somewhat subjective, for our purposes 'deep' is any kind of neural network in which there is more than one hidden layer. In this framework we have multiple hidden layers in which earlier layers in the network are tasked with detecting low-level details and as we progress through the network the layers are able to detect and learn more abstract features (we build abstract objects from objects which are less abstract).

We can see this in Figure 2.8 (reproduced from Zeiler and Fergus (2014)) in which the very first layer learns simple edge detectors and the last layer computes features akin to entire face detectors (as can be seen in the third row from the bottom in Layer 3).

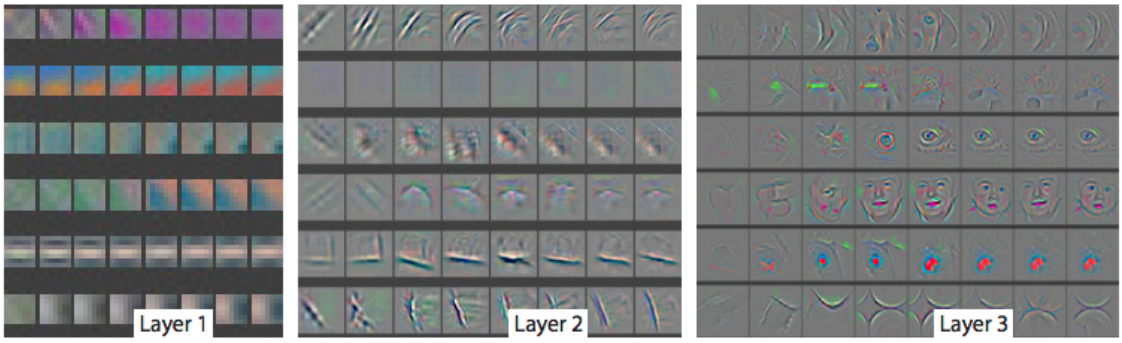


Figure 2.8 Filter visualisations of a convolutional neural network. Each successive layer learns a more highly abstract filter (e.g. Layer 1 has simple edge detectors, while Layer 3 has filters corresponding to face detectors). Each row shows the evolution of that particular filter (left = early training, right = late training). This figure was reproduced with permission from Zeiler and Fergus (2014).

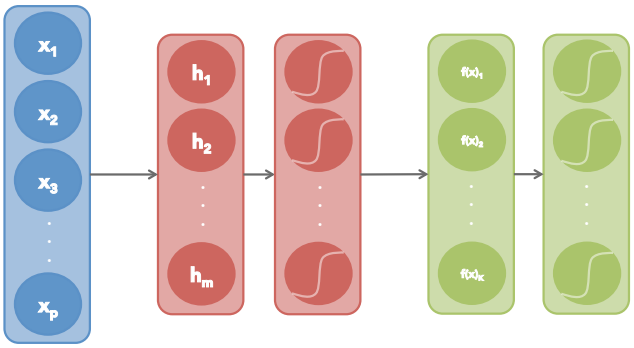


Figure 2.9 Layer representation of a multilayer perceptron. Compared to multinomial logistic regression (Figure 2.7) we now have a hidden layer (shown in red) $\mathbf{h} = \{\mathbf{h}_1, \ldots, \mathbf{h}_m\}$.

2.6 Convolutional neural networks

So far we have dealt with situations in which the input is comprised of p features. This basic assumption is quite applicable for many problems in machine learning, but when the input dimensionality is extremely large this can cause some problems. For example, if we look at the classic MNIST digit recognition dataset, we can see that each image is 28×28 pixels, which amounts to 784 pixels total. If we train an MLP with a hidden layer consisting of m units and a final layer of 10 units (since there are 10 classes), we will have in total a

dimensionality of:

$$|\mathbf{W}^{(1)}| + |\mathbf{b}^{(1)}| + |\mathbf{W}^{(2)}| + |\mathbf{b}^{(2)}| = (784 \times m) + m + (m \times 10) + 10$$

If $m = 128$ (i.e. we have 128 hidden units), this evaluates to 101,770. While this number of learnable parameters is by no means intractable by the standards of today's hardware, it is somewhat excessive to have a parameter assigned to every pixel in the original input image because it is not statistically efficient (nor is it computationally efficient). If we consider the way we identify objects as humans for example, if we were trying to find a specific word embedded somewhere in a page of text we would scan along the page systematically trying to find it. In this particular context, we could think of our visual cortex containing a 'filter' (one specific for our task at hand) which we 'apply' to each part of the text we focus on as we are scanning the page. We may even have filters at different granularities of the page, all the way from the individual strokes comprising a letter, to the letter itself, to even the entire word. We can see that the MLP has no notion of this kind of behaviour!

Let us override earlier notation and define $\mathbf{x}$ to be a 3-dimensional matrix (also called a *tensor*) with some dimension $(f \times h \times w)$, where f denotes the number of input channels (e.g. 3 for an RGB image) and h and w denote height and width, respectively. Instead of multiplying $\mathbf{x}$ with a matrix $\mathbf{W}$, we instead use a much smaller matrix $\mathbf{k} \in \mathbb{R}^{r \times r}$ (called a *kernel* or a *filter*, with some receptive field r) and slide this down and across $\mathbf{x}$, computing element-wise multiplications and sums to obtain a new channel (also called a *feature map*), $\mathbf{h} \in \mathbb{R}^{k \times h' \times w'}$, where k denotes the number of filters and $h' \leq h$ and $w' \leq w$. Concretely, this operation is called a *discrete convolution* where we are convolving the input $\mathbf{x}$ with kernel $\mathbf{k}$, which we denote as $\mathbf{x} * \mathbf{k}$. To illustrate an example of this, suppose our input and kernel are defined as such:

$$\mathbf{x} = \begin{bmatrix} 3 & 3 & 2 & 1 & 0 \\ 0 & 0 & 1 & 3 & 1 \\ 3 & 1 & 2 & 2 & 3 \\ 2 & 0 & 0 & 2 & 2 \\ 2 & 0 & 0 & 0 & 1 \end{bmatrix}, \ \mathbf{k} = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix} \tag{2.13}$$

Figure 2.10 shows the steps behind this operation. The blue matrix is $\mathbf{x}$, cyan matrix is the kernel $\mathbf{k}$, and we overlay the kernel at a particular location (highlighted in a darker blue), perform an element-wise multiplication of those locations with the corresponding elements of the kernel, and sum (i.e. a dot product, if we were to flatten the region in dark blue and dot product it with the flattened kernel). As seen in the figure, the result of this convolution

then becomes:

$$\mathbf{x} * \mathbf{k} = \begin{bmatrix} 12 & 12 & 17 \\ 10 & 17 & 19 \\ 9 & 6 & 14 \end{bmatrix} \tag{2.14}$$

In this example, because we slide the kernel in both the x and y directions one element at a time, this is referred to as a *stride* of 1.

Note that we can have multiple channels (feature maps) for a particular layer, in which case we will denote $\mathbf{h}_i$ as the i'th feature map. For a particular layer, the computation of the i'th feature map will be a result of convolving a separate kernel for every feature map in the preceding layer (note that if we are talking about the input layer $\mathbf{x}$ we will simply call them input channels). Concretely, we can then represent this computation as the following:

$$\mathbf{h}_j^{(l)} = g\Big(\sum_{i=1}^{n_{l-1}} \mathbf{k}_{ij}^{(l)} * \mathbf{h}_i^{(l-1)} \Big), \tag{2.15}$$

where $\mathbf{k}_{ij}^{(l)}$ denotes the particular kernel in the j'th feature map of the l'th layer corresponding to the i'th feature map (input channel) in the preceding layer, and n_{l-1} denotes the number of feature maps (input channels) in the preceding layer $l-1$.

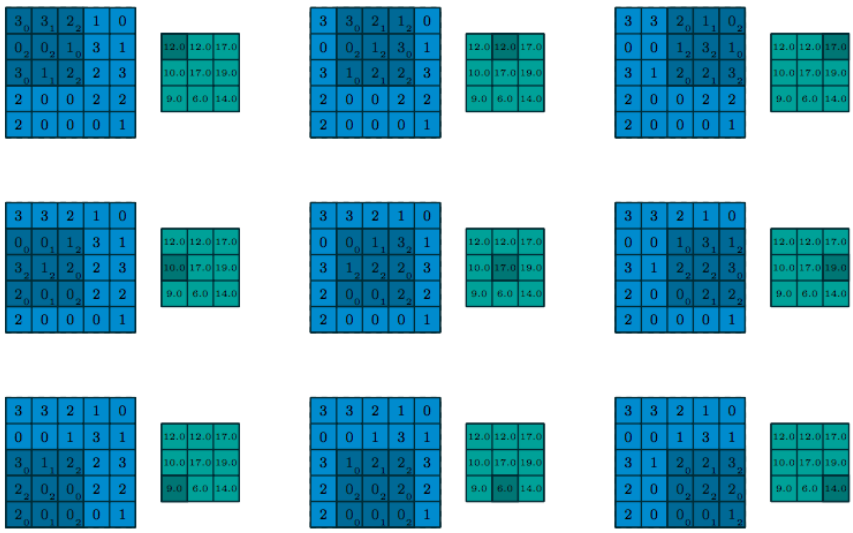


Figure 2.10 Example showing convolution between the input and kernel defined in Equation 2.13. This diagram was reproduced with permission from Dumoulin and Visin (2016).

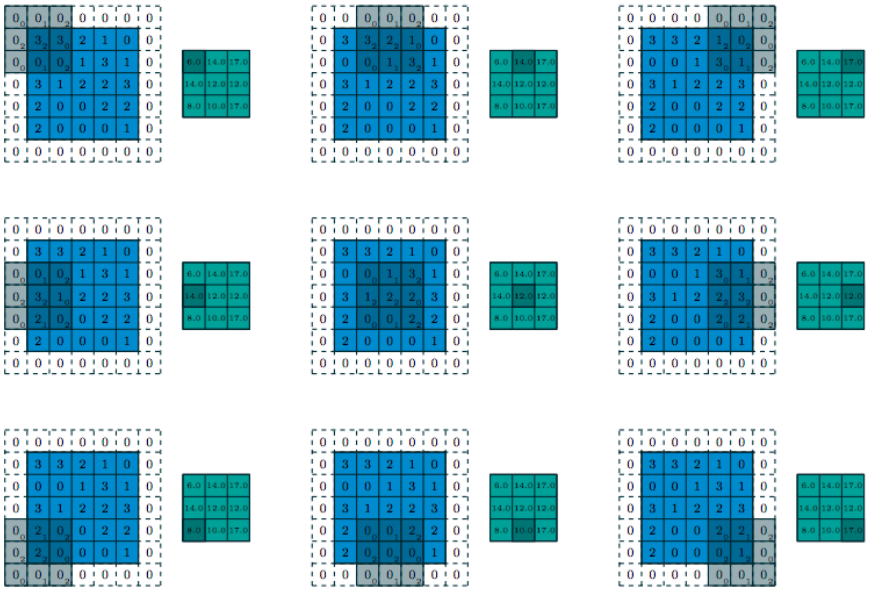


Figure 2.11 Example showing strided ($s = 2$) convolution between the input and kernel defined in Equation 2.13. The difference between this and Figure 2.10 is that we now shift the kernel by two elements when we shift across / down, and that the input has been padded by a border of zeros so that we can produce a 3×3 output.This diagram was reproduced with permission from Dumoulin and Visin (2016).

Convolutions are much, much more efficient in terms of both computational and statistical simplicity. For instance, if we took a $1 \times 28 \times 28$ image and convolved it with 64 kernels with a 3×3 receptive field using a stride of 2 and appropriate zero-padding, we would obtain a layer $\mathbf{h} \in \mathbb{R}^{64 \times 14 \times 14}$ with $64 \times 3 \times 3 = 576$ learnable parameters. Conversely, with an MLP, if we decided on a hidden layer with $14 \times 14 = 196$ layers the number of learnable parameters would be $|\mathbf{W}| = 784 \times 196 = 153,664$ parameters. This is clearly an enormous explosion in terms of the number of parameters, and it would be much easier to overfit with an MLP. From the point of view of statistical efficiency, because the same filter is slid across the entire region of the input image, we are enforcing the notion that the filter is a detector of some object and that this can exist anywhere in the image. Conversely, the MLP would assign different weights to different regions of the input image, which encourages the complete opposite notion. This means that with convolutions we earn a certain kind of invariance which is important, and that is *translational invariance*. Concretely, it means that if we are detecting some object of interest (say, a cat), it should not matter where it is in the input image because the filters are not tied to any specific location in the image.

Another type of invariance which would be desirable is the ability to be invariant to small

perturbations in the input. This could be achieved in several different ways but in deep learning a specific type of layer exists for this called a subsampling or pooling layer. Instead of convolving a kernel over the input however, we slide a function which takes as input the elements in its receptive field. For example, the most common layer of its type, the *max-pooling layer*, takes the maximum element in its receptive field. If we perform a 3×3 max-pooling for the input $\mathbf{x}$ in Equation 2.13 we obtain (stride = 1):

$$\text{maxpool}(\mathbf{x}, r = 3, s = 1) = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 2 & 3 \end{bmatrix} \tag{2.16}$$

This type of operation encourages invariance to small perturbations such as slight shifts in the input or minor changes to the contrast of the image.

With the two main building blocks of convolutional neural networks established (the convolutional layer and max-pooling layer), we can now examine an example architecture. We describe one of the simplest architectures, the 'LeNet', in Figure 2.12. In this, we have (grayscale) input image $\mathbf{x}$ with the first convolutional layer (labelled S1) consisting of 4 feature maps. The next layer, C1, is a subsampling layer which halves the dimension of both spatial dimensions. Afterwards, another convolution is performed resulting in siz feature maps, followed by another pooling layer. In order to make a classification, we take this hidden layer, flatten its elements into a single vector, and feed this into a fully-connected (i.e. MLP) layer. Because this is a fully-connected layer, we can think of this as being really an MLP which takes as input features extracted from the image (in the form of a bunch of low resolution feature maps summarising large regions of that image) and outputs a probability distribution over the classes we are trying to predict.

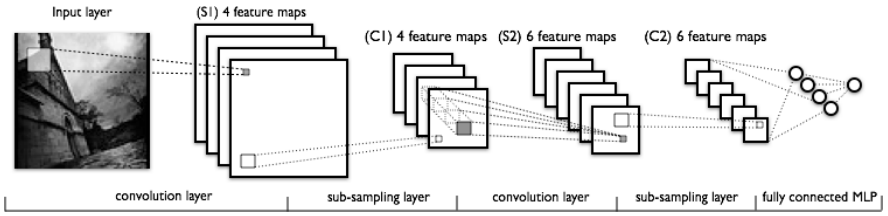


Figure 2.12 LeNet network. http://deeplearning.net/tutorial/lenet.html

Since the resurgence of convolutional neural networks, there have been several architectures proposed which are generally used as foundations to build upon. For example, one of the very first was AlexNet (Krizhevsky *et al.*, 2012), followed by 'VGGNet' (Simonyan and Zisserman,

2014) which proposed a simple way to increase the depth of convolutional nets without greatly blowing up the number of parameters needed to be learned. Currently, the most successful architecture has been residual networks (He *et al.*, 2015), which build on the VGG architecture by adding 'skip connections', which are alternate pathways that connect far-away layers. Not only does this stabilise network training, but allows one to construct deeper networks.

2.7 Autoencoders

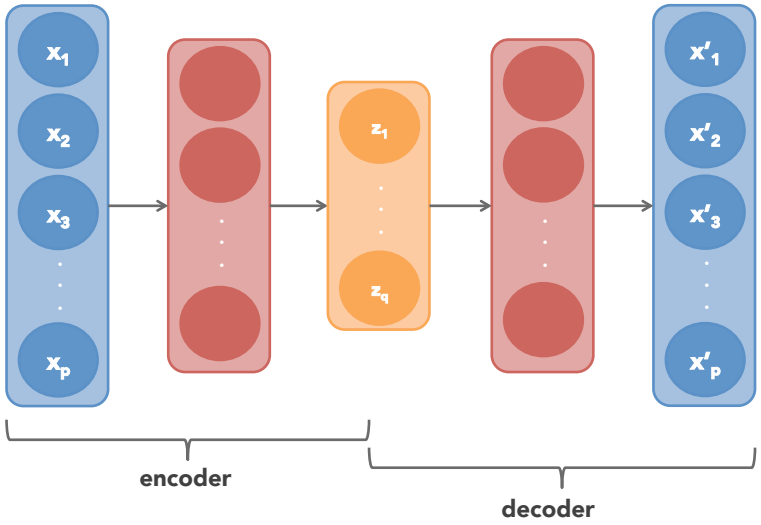


Figure 2.13 Layer representation of an autoencoder. In this we have an encoding function $f(\mathbf{x})$ which maps the input $\mathbf{x}$ to the 'bottleneck' layer $\mathbf{z}$. The corresponding decoding function, $g(\mathbf{x})$, tries to map back to the original input. In this example there is only one hidden layer for the encoding and decoding functions, but deeper autoencoders can be achieved via several hidden layers for both the encoding and decoding pathways.

Autoencoders are a class of neural network used for the purpose of unsupervised learning. Rather than try to predict some label $\mathbf{y}$ given some input $\mathbf{x}$, we try to reconstruct our input instead. This may sound rather uninteresting – surely a neural network with enough model capacity could simply learn the identity function to perfectly reconstruct the input – but this is not the objective. The autoencoder is designed in such a way that the hidden layers are (usually) of a smaller dimensionality than the input. This forces the network to learn a robust representation of its inputs, and to learn the factors of variation that actually matter in trying to reconstruct $\mathbf{x}$. Concretely, given some encoding function $f(\mathbf{x}) : \mathbb{R}^p \to \mathbb{R}^q$ and decoding function $g(\mathbf{z}) : \mathbb{R}^q \to \mathbb{R}^p$ (where $q << p$ usually), try to learn a 'good' encoding $\mathbf{z}$ (called a *bottleneck*) such that we minimise some distance between the original input $\mathbf{x}$

and its reconstruction $g(f(\mathbf{x}))$. This is illustrated in Figure 2.13, in which the encoding and decoding functions are MLPs consisting of one hidden layer each. While there are several different objectives one could use in the training of an autoencoder, we present the most commonly used one which is simply the squared L2 norm between the original input and its reconstruction:

$$\min_{\theta} L(\theta) = \frac{1}{n} \sum_{i=1}^{n} ||\mathbf{x}_i - g(f(\mathbf{x}_i))||_2^2 \tag{2.17}$$

While the relatively low dimensionality of the bottleneck layer $\mathbf{z}$ encourages the network to learn a robust representation of the input, another technique is to force the network to reconstruct $\mathbf{z}$ in the presence of noise. This is called the *denoising autoencoder* proposed by Vincent *et al.* (2008). In this formulation, we instead minimise the following loss function:

$$\min_{\theta} L(\theta) = \frac{1}{n} \sum_{i=1}^{n} ||\mathbf{x}_i - g(f(\mathbf{x}_i + \epsilon_i))||_2^2, \tag{2.18}$$

where ϵ_i could be sampled from some distribution such as $\mathcal{N}(0, 0.01)$, for example. We can see that the encoder is fed the corrupted version of the input $\mathbf{x} + \epsilon$ but the reconstruction is still based on the original input $\mathbf{x}$; this forces the network to learn what is actually useful and separate out the features that matter from noise.

It turns out autoencoders can be related back to a classic dimensionality reduction technique called PCA (principal components analysis). Bourlard and Kamp (1988) showed that the optimal parameters for a single hidden-layer MLP with a sigmoid nonlinearity is strongly related to its singular value decomposition. Therefore, we can look at autoencoders with multiple hidden layers as more powerful versions of PCA.

Before we wrap up this section it is important to mention that the type of architecture introduced in this section – the contracting (encoding) + contracting (decoding) pathway is useful in many different applications beyond autoencoding, such as segmentation or image translation, for instance. In that case, a convolutional autoencoder is used in which the fully-connected layers are replaced with convolutional ones instead.

2.8 Batch normalisation

It is common when training neural networks to perform some sort of scaling in the feature space to enable better training dynamics (LeCun *et al.*, 2012) and equal weighting of input features. For example, it is common to perform zero-mean unit variance scaling (ZMUV), in which for each feature we subtract its mean and divide by the variance over the training set. For images, it is common to compute the mean on the channel axis, and this can be done

either sample-wise (per image) or over the entire training set.

One may also wonder if performing this sort of scaling in the intermediate layers of the neural network will provide any benefits. It turns out this was explored by Ioffe and Szegedy (2015), in which they address an issue behind the training of deep networks called *internal covariate shift*. This term refers to the fact that when we are training the network, the inputs to a particular layer are affected by the parameters of the layers which come before it. Therefore, as we progressively go deeper into the network and examine a layer, the effect the preceding layers has on its distribution of inputs become more amplified. As it has been well-established that normalisation of input features results in faster convergence during training, the authors of batch normalisation propose to do the same thing but for the hidden layers of the network. Concretely, suppose $\mathbf{x} \in \mathbb{R}^{n \times p}$ is a minibatch of examples we wish to normalise, and $\mathbf{h}^{(l)} \in \mathbb{R}^{n \times m}$ denotes the l'th hidden layer with m units. Instead of performing the following computation $\mathbf{a}^{(l)} = g(\mathbf{a}^{(l-1)}\mathbf{W}^{(l)} + \mathbf{b}^{(l)})$, we perform the following steps:

$$\begin{aligned} \mathbf{z}^{(l-1)} &= \frac{\mathbf{a}^{(l-1)} - \mu(\mathbf{a}^{(l-1)})}{\epsilon + \sigma(\mathbf{a}^{(l-1)})} \text{ (normalise the input)} \\ \mathbf{a}^{(l)} &= g\Big((\mathbf{z}^{(l-1)}\boldsymbol{\gamma} + \boldsymbol{\beta})\mathbf{W}^{(l)} + \mathbf{b}^{(l)}\Big) \end{aligned} \tag{2.19}$$

In our case, $\mu(\cdot)$ and $\sigma(\cdot)$ compute the mean and variance on the feature axis, i.e., over all the other axes so that each hidden unit receives a mean and variance. In the case of images $\mathbf{x} \in \mathbb{R}^{n \times f \times h \times w}$ these statistics are computed only over the feature map axis (this means the statistics are computed over the batch axis and the spatial dimensions). Note that in the second line of Equation 2.19 we have introduced two extra parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$, whose purpose is, if the network deems it necessary during training, to 'undo' the effect of the batch norm. These parameters are vectors are both of length n_{l-1}.

2.9 Regularisation

Regularisation refers to a broad range of techniques in machine learning used to reduce the effects of overfitting. One common way to achieve this is through limiting the complexity of the model, which in turn makes it less likely to overfit. In neural networks, the way we do this is through introducing an extra term to the loss function which induces a high penalty when the model is more complex. For example, in L2 regularisation, we penalise the L2 norm of the network's weights, where the severity of the penalty is weighted by some hyperparameter λ:

$$\frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{x}_i, \mathbf{y}_i) + \lambda ||\boldsymbol{\theta}||_2^2 \tag{2.20}$$

There is also the L1 norm, which ends up enforcing sparsity in the network since its V-shaped function drives weights to zero, even if they are already small (see Figure 2.14).

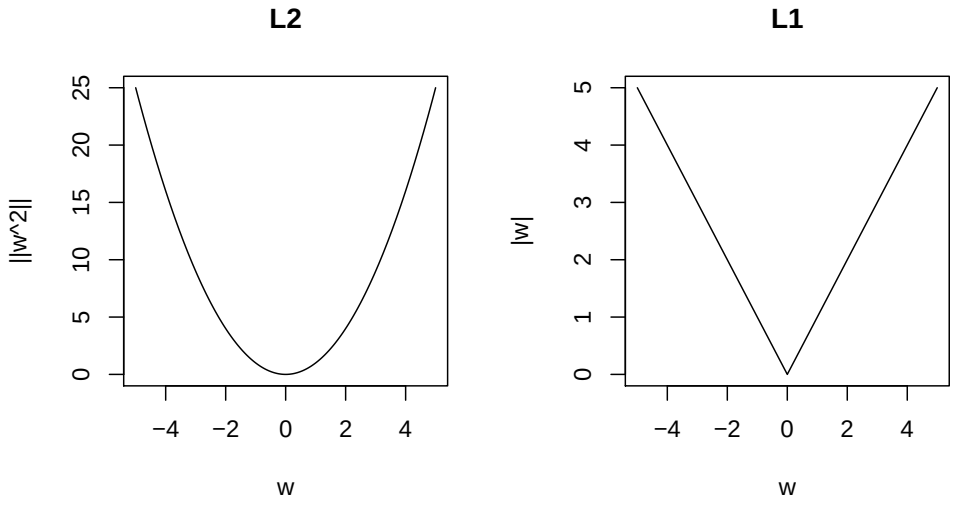


Figure 2.14 L2 and L1 loss functions for a scalar weight w. We can see that the V-shape of the L1 loss induces a sparsity constraint, since very small values of w induce a relatively higher penalty than that of L2, driving w toward zero.

Another technique which has seen popular use is dropout (Srivastava *et al.*, 2014). In this technique, we try to discourage neurons in the network from becoming codependent by randomly activating a subset of neurons at each layer during training (i.e., before the forward pass of each minibatch). Therefore, mathematically, we can view each neuron in the network as being a random variable drawn from a Bernoulli distribution, where it is 'on' with probability p and 'off' with probability $1 - p$. Therefore, to implement it, we simply multiply the output of a layer by a randomly sampled Bernoulli mask:

$$\mathbf{h}^{(l)} = \mathbf{B} \circ g(\mathbf{h}^{(l-1)}\mathbf{W} + \mathbf{b}^{(l)}) \tag{2.21}$$

During test time, to make a prediction, we either produce many different predictions and average them (since different subsets of neurons are turned off in each forward pass), or perform one pass by replacing $\mathbf{B}$ with the Bernoulli expectation of the neurons, which is $p(1-p)$. In terms of its regularisation effect, dropout can be seen as an implicit ensembling technique: rather than training multiple models, we instead train one model whose connectivity pattern is consistently changing during training. This can be seen as really training one large ensemble of models at once (where each model in the ensemble is to some degree smaller than

the original model with all neurons switched on). Therefore, as we mentioned earlier, at test time we can simply perform multiple forward passes through the network as if we had truly trained multiple models in our ensemble. Interestingly, this technique can also be used to quantify uncertainty in our predictions (Gal and Ghahramani, 2016), since we can simply perform multiple forward passes and compute both the mean and variance of our prediction.

In an ideal world, whenever a model is overfitting we could easily combat it by simply obtaining more data and training the model. Unfortunately, this often comes at a great cost. What we can do however is artificially increase the size of our training set. In deep learning this is commonly done – and often times very necessary – through a technique called 'data augmentation'. The different techniques one can use to do data augmentation largely depend on the dataset of interest. For example, in the case of digits classification, we can perform synthetic transformations such as performing random crops of the digit, minor additions of Gaussian noise, elastic transformations and so forth. However, unlike, say, in the case of natural images, horizontal and vertical flips would not make semantic sense in the context of digits, because flipping a '9' vertically would no longer make it a nine anymore. Therefore, we must make sure that the synthetic transformations we perform are *semantically meaningful*. In theory, one could also take this one step further and train a model – a generative model – to generate plausible synthetic examples from the data distribution, and we actually explore this in Chapter 4. For now however, we present an example of this in Figure 2.15 using a variational autoencoder (Kingma and Welling, 2013), in which we have learned a two-dimensional manifold of the MNIST digits. By walking this manifold we end up producing semantically meaningful interpolations between different digits.

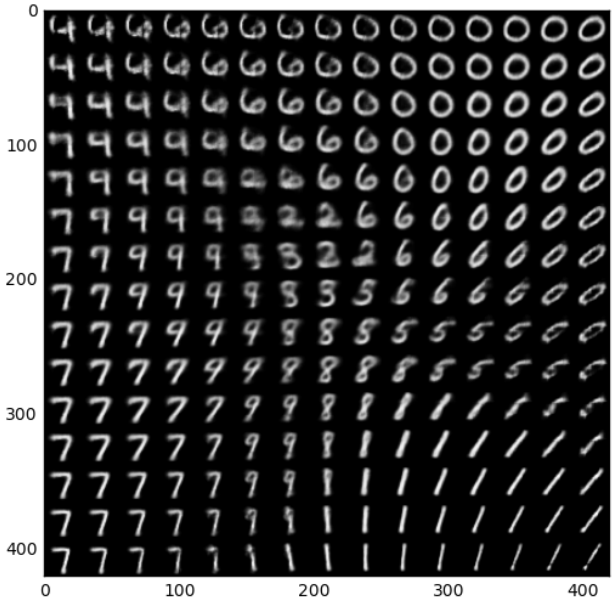


Figure 2.15 Variational autoencoder (VAE) trained on MNIST digits. In the VAE, the bottleneck layer encodes random variables which are constrained to be as close as possible to some prior distribution, which in our case is a two-dimensional isotropic Gaussian. Therefore, by sampling $\mathbf{z} \sim N(0, 1) \in \mathbb{R}^2$ and feeding these through the decoder we can obtain various plausible and artificially generated MNIST digits.

CHAPTER 3 ARTICLE 1: UNIMODAL PROBABILITY DISTRIBUTIONS FOR DEEP ORDINAL CLASSIFICATION

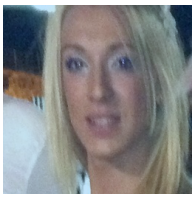
The contents of this chapter was accepted to ICML 2017 as a conference paper (Beckham and Pal, 2017). Due to original page limit constraints, we provide expanded discussion of this work in the appendix.

3.1 Introduction

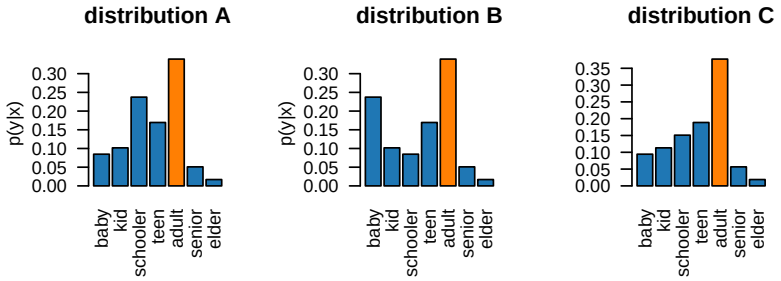
Ordinal classification (sometimes called ordinal regression) is a prediction task in which the classes to be predicted are discrete and ordered in some fashion. This is different from discrete classification in which the classes are not ordered, and different from regression in that we typically do not know the distances between the classes (unlike regression, in which we know the distances because the predictions lie on the real number line). Some examples of ordinal classification tasks include predicting the stages of disease for a cancer (Gentry *et al.*, 2015), predicting what star rating a user gave to a movie (Koren and Sill, 2011), or predicting the age of a person (Eidinger *et al.*, 2014).

Two of the easiest techniques used to deal with ordinal problems include either treating the problem as a discrete classification and minimising the cross-entropy loss, or treating the problem as a regression and using the squared error loss. The former ignores the inherent ordering between the classes, while the latter takes into account the distances between them (due to the square in the error term) but assumes that the labels are actually real-valued – that is, adjacent classes are equally distant. Furthermore, the cross-entropy loss – under a one-hot target encoding – is formulated such that it only 'cares' about the ground truth class, and that probability estimates corresponding to the other classes may not necessarily make sense in context. We present an example of this in Figure 3.1, showing three probability distributions: A, B, and C, all conditioned on some input image. Highlighted in orange is the ground truth (i.e. the image is of an adult), and all probability distributions have identical cross-entropy: this is because the loss only takes into account the ground truth class, $-\log(p(y|\mathbf{x})_c)$, where $c = adult$, and all three distributions have the same probability mass for the adult class.

Despite all distributions having the same cross-entropy loss, some distributions are 'better' than others. For example, between A and B, A is preferred, since B puts an unusually high mass on the baby class. However, A and B are both unusual, because the probability mass



(a) An adult woman



(b) Three probability distributions exhibiting the same mass for the 'adult' class and therefore the same cross-entropy error.

Figure 3.1 Three ordinal probability distributions conditioned on an image of an adult woman. Distributions A and B are unusual in the sense that they are multi-modal.

does not gradually decrease to the left and right of the ground truth. In other words, it seems unusual to place more confidence on 'schooler' than 'teen' (distribution A) considering that a teenager looks more like an adult than a schooler, and it seems unusual to place more confidence on 'baby' than 'teen' considering that again, a teenager looks more like an adult than a baby. Distribution C makes the most sense because the probability mass gradually decreases as we move further away from the most confident class. In this paper, we propose a simple method to enforce this constraint, utilising the probability mass function of either the Poisson or binomial distribution.

For the remainder of this paper, we will refer to distributions like C as 'unimodal' distributions; that is, distributions where the probability mass gradually decreases on both sides of the class that has the majority of the mass.

3.1.1 Related work

Our work is inspired by the recent work of Hou *et al.* (2016), who shed light on the issues associated with different probability distributions having the same cross-entropy loss for ordinal problems. In their work, they alleviate this issue by minimising the 'Earth mover's distance', which is defined as the minimum cost needed to transform one probability distribution to another. Because this metric takes into account the distances between classes – moving probability mass to a far-away class incurs a large cost – the metric is appropriate to minimise for an ordinal problem. It turns out that in the case of an ordinal problem, the

Earth mover's distance reduces down to Mallow's distance:

$$\text{emd}(\hat{\mathbf{y}}, \mathbf{y}) = \left(\frac{1}{K}\right)^{\frac{1}{l}} ||\text{cmf}(\hat{\mathbf{y}}) - \text{cmf}(\mathbf{y})||_l, \tag{3.1}$$

where $\text{cmf}(\cdot)$ denotes the cumulative mass function for some probability distribution, $\mathbf{y}$ denotes the ground truth (one-hot encoded), $\hat{\mathbf{y}}$ the corresponding predicted probability distribution, and K the number of classes. The authors evaluate the EMD loss on two age estimation and one aesthetic estimation dataset and obtain state-of-the-art results. However, the authors do not show comparisons between the probability distributions learned between EMD and cross-entropy.

Unimodality has been explored for ordinal neural networks in da Costa *et al.* (2008). They explored the use of the binomial and Poisson distributions and a non-parametric way of enforcing unimodal probability distributions (which we do not explore). One key difference between their work and ours here is that we evaluate these unimodal distributions in the context of deep learning, where the datasets are generally much larger and have more variability; however, there are numerous other differences which we will highlight throughout this paper.

Beckham and Pal (2016) explored a loss function with an intermediate form between a cross-entropy and regression loss. In their work the squared error loss is still used, but a probability distribution over classes is still learned. This is done by adding a regression layer (i.e. a one-unit layer) at the top of what would normally be the classification layer, $p(y|\mathbf{x})$. Instead of learning the weight vector $\mathbf{a}$ it is fixed to $[0, \dots, K-1]^T$ and the squared error loss is minimised. This can be interpreted as drawing the class label from a Gaussian distribution $p(c|\mathbf{x}) = N(c; \mathbb{E}[\mathbf{a}]_{p(y|\mathbf{x})}, \sigma^2)$. This technique was evaluated against the diabetic retinopathy dataset and beat most of the baselines employed. Interestingly, since $p(c|\mathbf{x})$ is a Gaussian, this is also unimodal, though it is a somewhat odd formulation as it assumes c is continuous when it is really discrete.

Cheng (2007) proposed the use of binary cross-entropy or squared error on an ordinal encoding scheme rather than the one-hot encoding which is commonly used in discrete classification. For example, if we have K classes, then we have labels of length $K-1$, where the first class is $[0, \dots, 0]$, second class is $[1, \dots, 0]$, third class is $[1, 1, \dots, 0]$ and so forth. With this formulation, we can think of the i'th output unit as computing the cumulative probability $p(y > i|\mathbf{x})$, where $i \in \{0, \dots, K-2\}$. Frank and Hall (2001) also proposed this scheme but in a more general sense by using multiple classifiers (not just neural networks) to model each cumulative probability, and Niu *et al.* (2016) proposed a similar scheme using CNNs for age

estimation. This technique however suffers from the issue that the cumulative probabilities $p(y > 0 \mid \mathbf{x}), \ldots, p(y > K-2 \mid \mathbf{x})$ are not guaranteed to be monotonically decreasing, which means that if we compute the discrete probabilities $p(y = 0 \mid \mathbf{x}), \ldots, p(y = K-1 \mid \mathbf{x})$ these are not guaranteed to be strictly positive. To address the monotonicity issue, Schapire *et al.* (2002) proposed a heuristic solution.

There are other ordinal techniques but which do not impose unimodal constraints. The proportional odds model (POM) and its neural network extensions (POMNN, CHNN (Gutiérrez *et al.*, 2014)) do not suffer from the monotonicity issue due to the utilization of monotonically increasing biases in the calculation of probabilities. The stick-breaking approach by Khan *et al.* (2012), which is a reformulation of the multinomial logit (softmax), could also be used in the ordinal case as it technically imposes an ordering on classes.

3.1.2 Poisson distribution

The Poisson distribution is commonly used to model the probability of the number of events, $k \in \mathbb{N} \cup 0$ occurring in a particular interval of time. The average frequency of these events is denoted by $\lambda \in R^+$. The probability mass function is defined as:

$$p(k; \lambda) = \frac{\lambda^k \exp(-\lambda)}{k!}, \tag{3.2}$$

where $0 \leq k \leq K-1$. While we are not actually using this distribution to model the occurrence of events, we can make use of its probability mass function (PMF) to enforce discrete unimodal probability distributions. For a purely technical reason, we instead deal with the log of the PMF:

$$\begin{aligned} \log\left[\frac{\lambda^k exp(-\lambda)}{k!}\right] &= \log(\lambda^k exp(-\lambda)) - \log(k!) \\ &= \log(\lambda^k) + \log(exp(-\lambda)) - \log(k!) \\ &= k\log(\lambda) - \lambda - \log(k!). \end{aligned} \tag{3.3}$$

If we let $f(\mathbf{x})$ denote the scalar output of our deep net (where $f(\mathbf{x}) > 0$ which can be enforced with the softplus nonlinearity), then we denote $h(\mathbf{x})_j$ to be:

$$j\log(f(\mathbf{x})) - f(\mathbf{x}) - \log(j!), \tag{3.4}$$

where we have simply replaced the λ in equation (3.3) with $f(\mathbf{x})$. Then, $p(y = j|\mathbf{x})$ is simply a softmax over $h(\mathbf{x})$:

$$p(y = j|\mathbf{x}) = \frac{\exp(-h(\mathbf{x})_j/\tau)}{\sum_{i=1}^{K} \exp(-h(\mathbf{x})_i/\tau)}, \tag{3.5}$$

which is required since the support of the Poisson is infinite. We have also introduced a hyperparameter to the softmax, τ, to control the relative magnitudes of each value of $p(y = j|\mathbf{x})$ (i.e., the variance of the distribution). Note that as $\tau \to \infty$, the probability distribution becomes more uniform, and as $\tau \to 0$, the distribution becomes more 'one-hot' like with respect to the class with the largest pre-softmax value. We can illustrate this technique in terms of the layers at the end of the deep network, which is shown in Figure 3.2.

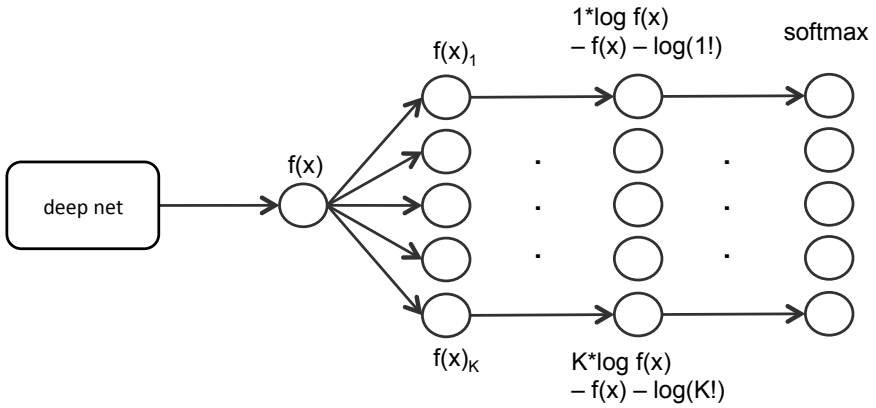


Figure 3.2 The first layer after $f(\mathbf{x})$ is a 'copy' layer, that is, $f(\mathbf{x}) = f(\mathbf{x})_1 = \cdots = f(\mathbf{x})_K$. The second layer applies the log Poisson PMF transform followed by the softmax layer.

We note that the term in equation (3.4) can be re-arranged and simplified to

$$\begin{aligned} h(\mathbf{x})_j &= j \ \log(f(\mathbf{x})) - f(\mathbf{x}) - \log(j!) \\ &= -f(\mathbf{x}) + j \log(f(\mathbf{x})) - \log(j!) \\ &= -f(\mathbf{x}) + b_j(f(\mathbf{x})). \end{aligned} \tag{3.6}$$

In this form, we can see that the probability of class j is determined by the scalar term $f(\mathbf{x})$ and a bias term that also depends on $f(\mathbf{x})$. Another technique that uses biases to determine class probabilities is the proportional odds model (POM), also called the ordered logit (McCullagh, 1980), where the cumulative probability of a class depends on a learned bias:

$$p(y \leq j \mid \mathbf{x}) = \text{sigm}(f(\mathbf{x}) - \mathbf{b}_j), \tag{3.7}$$

where $\mathbf{b}_1 < \cdots < \mathbf{b}_K$. Unlike our technique however, the bias vector $\mathbf{b}$ is not a function of $\mathbf{x}$

nor $f(\mathbf{x})$, but a fixed vector that is learned during training, which is interesting. Furthermore, probability distributions computed using this technique are not guaranteed to be unimodal.

Figure 3.3 shows the resulting probability distributions for values of $f(x) \in [0.1, 4.85]$ when $\tau = 1.0$ and $\tau = 0.3$. We can see that all distributions are unimodal and that by gradually increasing $f(\mathbf{x})$ we gradually change which class has the most mass associated with itself. The τ is also an important parameter to tune as it alters the variance of the distribution. For example, in Figure 3.3(a), we can see that if we are confident in predicting the second class, $f(\mathbf{x})$ should be ~ 2.6, though in this case the other classes receive almost just as much probability mass. If we set $\tau = 0.3$ however (Figure 3.3(b)), at $f(\mathbf{x}) = 2.6$ the second class has relatively more mass, which is to say we are even more confident that this is the correct class. An unfortunate side effect of using the Poisson distribution is that the variance is equivalent to the mean, λ. This means that in the case of a large number of classes probability mass will be widely distributed, and this can be seen in the $K = 8$ case in Figure 3.4. While careful selection of τ can mitigate this, we also use this problem to motivate the use of the binomial distribution.

In the work of da Costa *et al.* (2008), they address the infinite support problem by using a 'right-truncated' Poisson distribution. In this formulation, they simply find the normalization constant such that the probabilities sum to one. This is almost equivalent to what we do, since we use a softmax, although the softmax exponentiates its inputs and we also introduce the temperature parameter τ to control for the variance of the distribution.

3.1.3 Binomial distribution

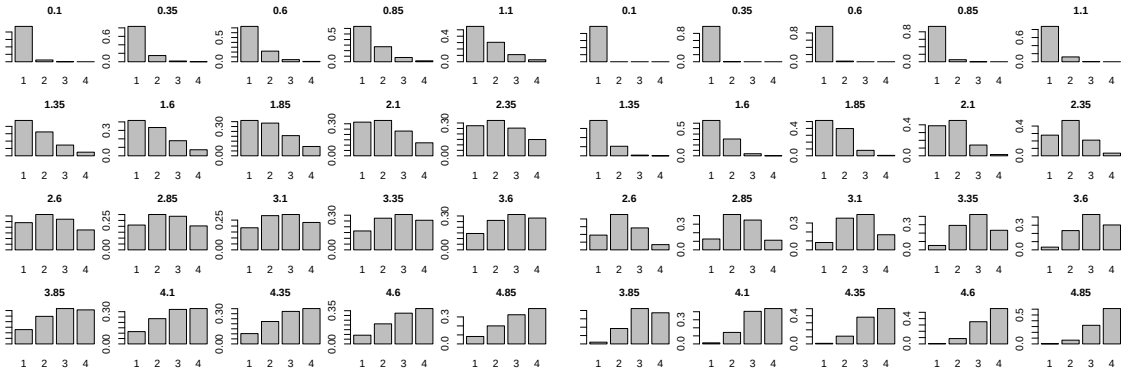
The binomial distribution is used to model the probability of a given number of 'successes' out of a given number of trials and some success probability. The probability mass function for this distribution – for k successes (where $0 \leq k \leq K - 1$), given $K - 1$ trials and success probability p – is:

$$p(k; K - 1, p) = \binom{K-1}{k} p^k (1 - p)^{K-1-k} \tag{3.8}$$

In the context of applying this to a neural network, k denotes the class we wish to predict, $K - 1$ denotes the number of classes (minus one since we index from zero), and $p = f(\mathbf{x}) \in [0, 1]$ is the output of the network that we wish to estimate. While no normalisation is theoretically needed since the binomial distribution's support is finite, we still had to take the log of the PMF and normalise with a softmax to address numeric stability issues. This means the resulting network is equivalent to that shown in Figure 3.2, but with the log binomial PMF instead of Poisson. Just like with the Poisson formulation, we can introduce

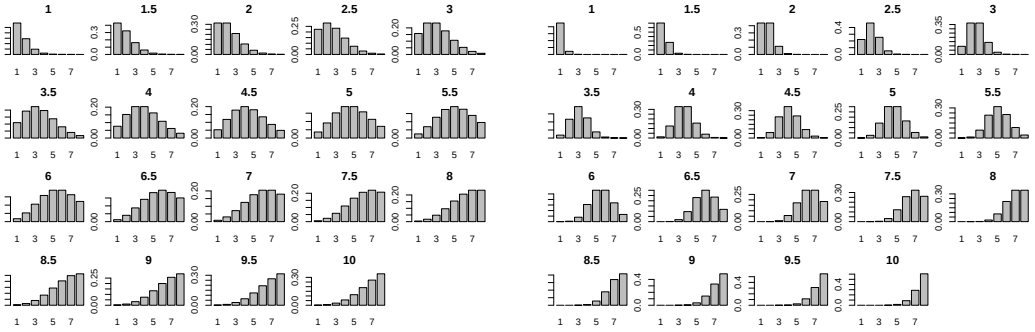
the temperature term τ into the resulting softmax to control for the variance of the resulting distribution.

Figure 3.5 shows the resulting distributions achieved by varying p for when $K = 4$ and $K = 8$.



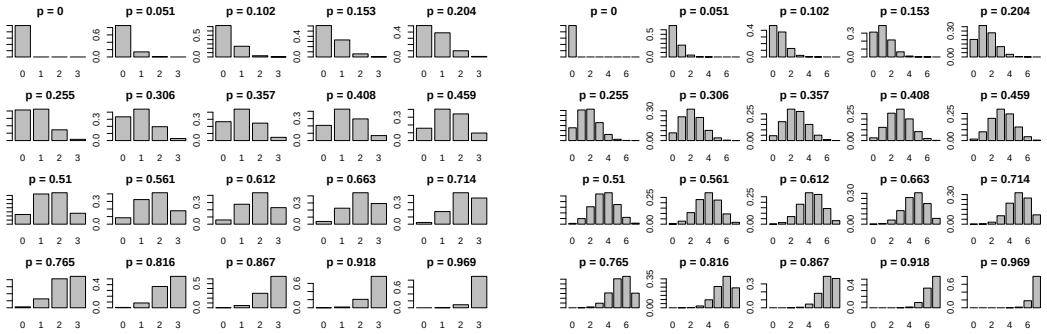
(a) $K = 4, \tau = 1.0$ (b) $K = 4, \tau = 0.3$

Figure 3.3 Illustration of the probability distributions that are obtained from varying $f(\mathbf{x}) \in [0.1, 4.85]$ for when there are four classes ($K = 4$) and when $\tau = 1.0$ (left) and $\tau = 0.3$ (right). We can see that lowering τ results in a lower variance distribution. Depending on the number of classes, it may be necessary to tune τ to ensure the right amount of probability mass hits the correct class.



(a) $K = 8, \tau = 1.0$ (b) $K = 8, \tau = 0.3$

Figure 3.4 Illustration of the probability distributions that are obtained from varying $f(\mathbf{x}) \in [1, 10]$ for when there are eight classes ($K = 8$) and when $\tau = 1.0$ (left) and $\tau = 0.3$ (right).



(a) $K = 4$ (b) $K = 8$

Figure 3.5 Illustration of the probability distributions that are obtained from varying $p \in [0, 1]$ for the binomial classes when $K = 4$ (left) and $K = 8$ (right).

3.2 Methods and Results

In this section we go into details of our experiments, including the datasets used and the precise architectures.

3.2.1 Data

We make use of two ordinal datasets appropriate for deep neural networks:

- Diabetic retinopathy[1]. This is a dataset consisting of extremely high-resolution fundus image data. The training set consists of 17,563 pairs of images (where a pair consists of a left and right eye image corresponding to a patient). In this dataset, we try and predict from five levels of diabetic retinopathy: no DR (25,810 images), mild DR (2,443 images), moderate DR (5,292 images), severe DR (873 images), or proliferative DR (708 images). A validation set is set aside, consisting of 10% of the patients in the training set. The images are pre-processed using the technique proposed by competition winner Graham (2015) and subsequently resized to 256px width and height.

- The Adience face dataset[2] (Eidinger *et al.*, 2014). This dataset consists of 26,580 faces belonging to 2,284 subjects. We use the form of the dataset where faces have been pre-cropped and aligned. We further pre-process the dataset so that the images are 256px in width and height. The training set consists of merging the first four cross-validation

[1]https://www.kaggle.com/c/diabetic-retinopathy-detection/
[2]http://www.openu.ac.il/home/hassner/Adience/data.html

folds together (the last cross-validation fold is the test set), which comprises a total of 15,554 images. From this, 10% of the images are held out as part of a validation set.

3.2.2 Network

We make use of a modest ResNet (He *et al.*, 2015) architecture to conduct our experiments. Table 3.1 describes the exact architecture. We use the ReLU nonlinearity and HeNormal initialization throughout the network.

Table 3.1 Description of the ResNet architecture used in the experiments. For convolution, WxH@FsS = filter size of dimension W x H, with F feature maps, and a stride of S. For average pooling, WxHsS = a pool size of dimension W x H with a stride of S. This architecture comprises a total of 4,307,840 learnable parameters.

Layer	**Output size**
Input (3x224x224)	3 x 224 x 224
Conv (7x7@32s2)	32 x 112 x 112
MaxPool (3x3s2)	32 x 55 x 55
2 x ResBlock (3x3@64s1)	32 x 55 x 55
1 x ResBlock (3x3@128s2)	64 x 28 x 28
2 x ResBlock (3x3@128s1)	64 x 28 x 28
1 x ResBlock (3x3@256s2)	128 x 14 x 14
2 x ResBlock (3x3@256s1)	128 x 14 x 14
1 x ResBlock (3x3@512s2)	256 x 7 x 7
2 x ResBlock (3x3@512s1)	256 x 7 x 7
AveragePool (7x7s7)	256 x 1 x 1

We conduct the following experiments for both DR and Adience datasets:

- (Baseline) cross-entropy loss. This simply corresponds to a softmax layer for K classes at the end of the average pooling layer in Table 3.1. For Adience and DR, this corresponds to a network with 4,309,896 and 4,309,125 learnable parameters, respectively.
- (Baseline) squared-error loss. Rather than regress $f(\mathbf{x})$ against y, we regress with $(K-1)sigm(f(\mathbf{x}))$, since we have observed better results with this formulation in the past. For Adience and DR, this corresponds t 4,309,905 and 4,309,131 learnable parameters, respectively.

- Cross-entropy loss using the Poisson and binomial extensions at the end of the architecture (see Figure 3.2). For Adience and DR, this corresponds to 4,308,097 learnable parameters for both. Although da Costa *et al.* (2008) mention that cross-entropy or squared error can be used, their equations assume a squared error between the (one-hot encoded) ground truth and $p(y|\mathbf{x})$, whereas we use cross-entropy.

- EMD loss (equation 3.1) where $\ell = 2$ (i.e. Euclidean norm) and the entire term is squared (to get rid of the square root induced by the norm) using Poisson and binomial extensions at the end of architecture. Again, this corresponds to 4,308,097 learnable parameters for both networks.

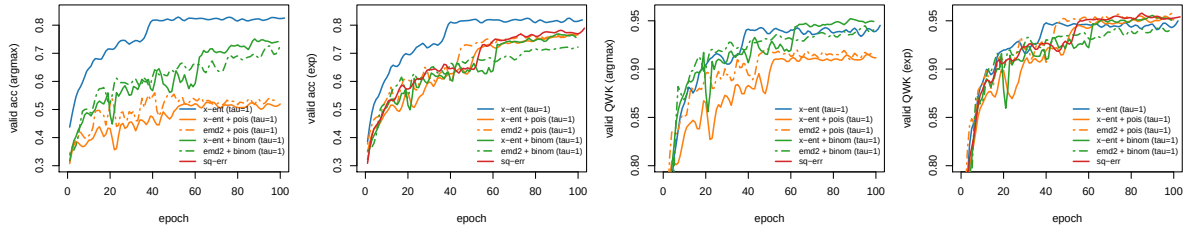
Amongst these experiments, we use $\tau = 1$ and also learn τ as a bias. When we learn τ, we instead learn $\text{sigm}(\tau)$ since we found this made training more stable. Note that we can also go one step further and learn τ as a function of $\mathbf{x}$, though experiments did not show any significant gain over simply learning it as a bias. However, one advantage of this technique is that the network can quantify its uncertainty on a per-example basis. It is also worth noting that the Poisson and binomial formulations are slightly underparameterised compared to their baselines, but experiments we ran that addressed this (by matching model capacity) did not yield significantly different results.

It is also important to note that in the case of ordinal prediction, there are two ways to compute the final prediction: simply taking the argmax of $p(y|\mathbf{x})$ (which is what is simply done in discrete classification), or taking a 'smoothed' prediction which is simply the expectation of the integer labels w.r.t. the probability distribution, i.e., $\mathbb{E}[0, \ldots, K-1]_{p(y|x)}$. For the latter, we call this the 'expectation trick'. A benefit of the latter is that it computes a prediction that considers the probability mass of all classes. One benefit of the former however is that we can use it to easily rank our predictions, which can be important if we are interested in computing top-k accuracy (rather than top-1).

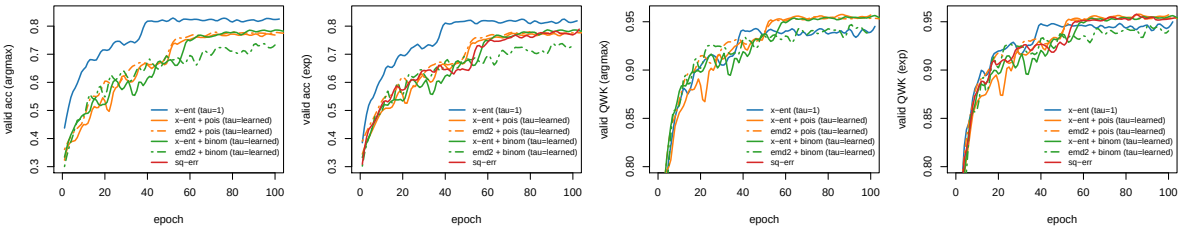
We also introduce an ordinal evaluation metric – the quadratic weighted kappa (QWK) (Cohen, 1968) – which has seen recent use on ordinal competitions on Kaggle. Intuitively, this is a number between [-1,1], where a kappa $\kappa = 0$ denotes the model does no better than random chance, $\kappa < 0$ denotes worst than random chance, and $\kappa > 0$ better than random chance (with $\kappa = 1$ being the best score). The 'quadratic' part of the metric imposes a quadratic penalty on misclassifications, making it an appropriate metric to use for ordinal problems.[3]

[3]The quadratic penalty is arbitrary but somewhat appropriate for ordinal problems. One can plug in any cost matrix into the kappa calculation.

All experiments utilise an ℓ_2 norm of 10^{-4}, ADAM optimiser (Kingma and Ba, 2014) with initial learning rate 10^{-3}, and batch size 128. A 'manual' learning rate schedule is employed where we manually divide the learning rate by 10 when either the validation loss or valid set QWK plateaus (whichever plateaus last) down to a minimum of 10^{-4} for Adience and 10^{-5} for DR.[4]



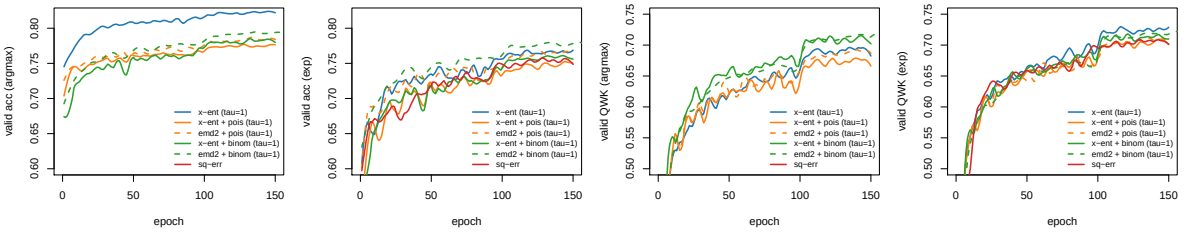
(a) Learning curves for Adience dataset, for $\tau = 1.0$. For both accuracy and QWK, both the argmax and expectation way of computing a prediction are employed.



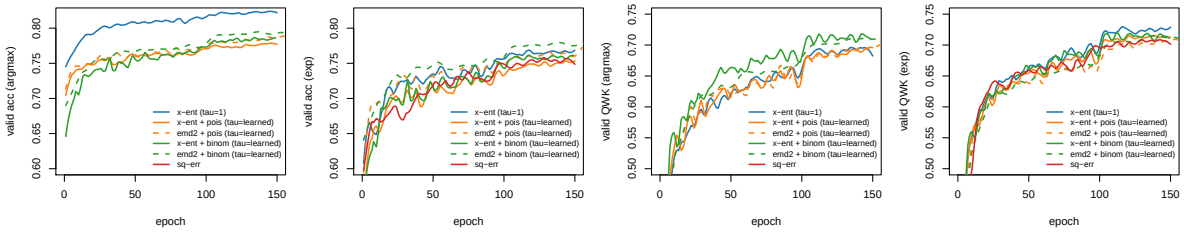
(b) Learning curves for Adience dataset, for when τ is learned as a bias. For both accuracy and QWK, both the argmax and expectation way of computing a prediction are employed.

Figure 3.6 Experiments for the Adience dataset. For $\tau = 1$ and $\tau =$ learned, we compare typical cross-entropy loss (blue), cross-entropy/EMD with Poisson formulation (orange solid / dashed, respectively), cross-entropy/EMD with binomial formulation (green solid / dashed, respectively), and regression (red). Learning curves have been smoothed with a LOESS regression for presentation purposes.

[4]We also re-ran experiments using an automatic heuristic to change the learning rate, and similar experimental results were obtained.

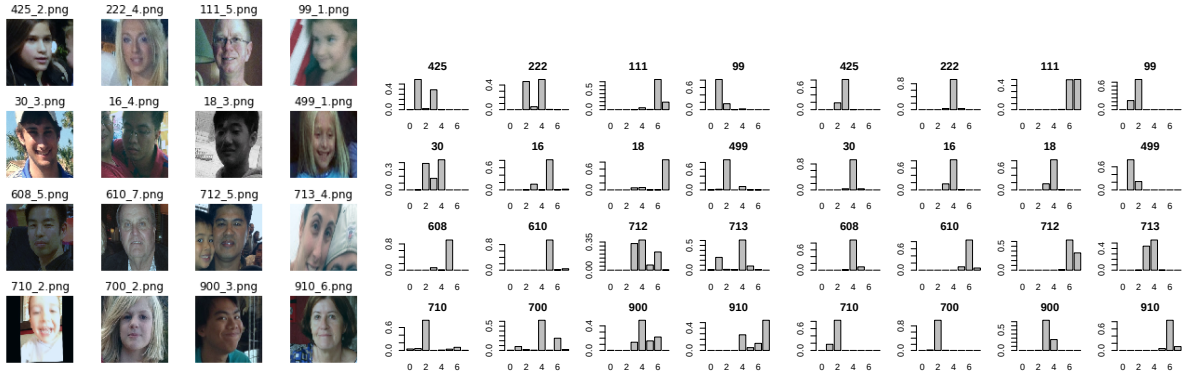


(a) Learning curves for diabetic retinopathy dataset, for $\tau = 1.0$. For both accuracy and QWK, both the argmax and expectation way of computing a prediction are employed.



(b) Learning curves for diabetic retinopathy dataset, where τ is made a learnable bias. For both accuracy and QWK, both the argmax and expectation way of computing a prediction are employed.

Figure 3.7 Experiments for the diabetic retinopathy (DR) dataset. For $\tau = 1$ and $\tau =$ learned, we compare typical cross-entropy loss (blue), cross-entropy/EMD with Poisson formulation (orange solid / dashed, respectively), cross-entropy/EMD with binomial formulation (green solid / dashed, respectively), and regression (red). Learning curves have been smoothed with a LOESS regression for presentation purposes.



(a) Faces from Adience valid set (b) Cross-entropy (baseline) (c) Cross-entropy + Poisson ($\tau = 1.0$)

Figure 3.8 Probability distributions over selected examples in the validation set for Adience (those selected have non-unimodal probability distributions for the cross-entropy baseline). Left: from cross-entropy + Poisson model (τ learned), right: cross-entropy (baseline) model

3.2.3 Experiments

Figure 3.6 shows the experiments run for the Adience dataset, for when $\tau = 1.0$ (Figure 3.6(a)) and when τ is learned (Figure 3.6(b)). We can see that for our methods, careful selection of τ is necessary for the accuracy on the validation set to be on par with that of the cross-entropy baseline. For $\tau = 1.0$, accuracy is poor, but even less so when τ is learned. To some extent, using the smoothed prediction with the expectation trick alleviates this gap. However, because the dataset is ordinal, accuracy can be very misleading, so we should also consider the QWK. For both argmax and expectation, our methods either outperform or are quite competitive with the baselines, with the exception of the QWK argmax plot for when $\tau = 1$, where only our binomial formulations were competitive with the cross-entropy baseline. Overall, considering all plots in Figure 3.6 it appears the binomial formulation produces better results than Poisson. There also appears to be some benefit gained from using the EMD loss for Poisson, but not for binomial.

Figure 3.7 show the experiments run for diabetic retinopathy. We note that unlike Adience, the validation accuracy does not appear to be so affected across all specifications of τ. One potential reason for this is due to Adience having a larger number of classes compared to DR. As we mentioned earlier, the Poisson distribution is somewhat awkward as its variance is equivalent to its mean. Since most of the probability mass sits at the mean, if the mean of the distribution is very high (which is the case for datasets with a large K such as Adience), then the large variance can negatively impact the distribution by taking probability mass away from the correct class. We can see this effect by comparing the distributions in Figure 3.3 ($k = 4$) and Figure 3.4 ($k = 8$). As with the Adience dataset, the use of the expectation trick brings the accuracy of our methods to be almost on-par with the baselines. In terms of QWK, only our binomial formulations appear to be competitive, but only in the argmax case do one of our methods (the binomial formulation) beat the cross-entropy baseline. At least for accuracy, there appears to be some gain in using the EMD loss for the binomial formulation. Because DR is a much larger dataset compared to Adience, it is possible that the deep net is able to learn reasonable and 'unimodal-like' probability distributions without it being enforced in the model architecture.

Overall, across both datasets the QWK for our methods are generally at least competitive with the baselines, especially if we learn τ to control for the variance. In the empirical results of da Costa *et al.* (2008), they found that the binomial formulation performed better than the Poisson, and when we consider all of our results in Figure 3.6 and 3.7 we come to the same conclusion. They justify this result by defining the 'flexibility' of a discrete probability distribution and show that the binomial distribution is more 'flexible' than Poisson. From

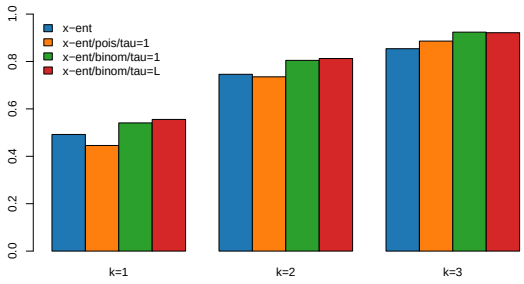


Figure 3.9 Top-k accuracies computed on the Adience test set, where $k \in \{1, 2, 3\}$.

our results, we believe that these unimodal methods act as a form of regularization which can be useful in regimes where one is interested in top-k accuracy. For example, in the case of top-k accuracy, we want to know if the ground truth was in the top k predictions, and we may be interested in such metrics if it is difficult to achieve good top-1 accuracy. Assume that our probability distribution $p(y|\mathbf{x})$ has most of its mass on the wrong class, but the correct class is on either side of it. Under a unimodal constraint, it is guaranteed that the two classes on either side of the majority class will receive the next greatest amount of probability mass, and this can result in a correct prediction if we consider top-2 or top-3 accuracy. To illustrate this, we compute the top-k accuracy on the test set of the Adience dataset, shown in Figure 3.9. We can see that even with the worst-performing model – the Poisson formulation with $\tau = 1$ (orange) – produces a better top-3 accuracy than the cross-entropy baseline (blue).

3.3 Conclusion

In conclusion, we present a simple technique to enforce unimodal ordinal probabilistic predictions through the use of the binomial and Poisson distributions. This is an important property to consider in ordinal classification because of the inherent ordering between classes. We evaluate our technique on two ordinal image datasets and obtain results competitive or superior to the cross-entropy baseline for both the quadratic weighted kappa (QWK) metric and top-k accuracy for both cross-entropy and EMD losses, especially under the binomial distribution. Lastly, the unimodal constraint can makes the probability distributions behave more sensibly in certain settings. However, there may be ordinal problems where a multimodal distribution may be more appropriate. We leave an exploration of this issue for future work. Code will be made available here.[5]

[5]https://github.com/christopher-beckham/deep-unimodal-ordinal

3.4 Acknowledgements

We thank Samsung for funding this research. We would like to thank the contributors of Theano (Theano Development Team, 2016) and Lasagne (Dieleman *et al.*, 2015) (which this project was developed in predominantly), as well as Keras (Chollet *et al.*, 2015) for extra useful code. We thank the ICML reviewers for useful feedback, as well as Eibe Frank.

CHAPTER 4 GENERATIVE ADVERSARIAL NETWORKS

Unsupervised learning is a branch of machine learning which has attracted great interest recently, due to the abundance of unlabeled data and the cost in acquiring labels for them. Because of this, we are highly interested in devising techniques which are able to extract useful bits of information from this data, hopefully providing a somewhat reasonable alternative to simply acquiring more labels. A common situation is where one has a large corpus of data they wish to use for classification, but only a very small percentage of it is labeled. Naively, one could simply train a classifier on only the labeled data and discard the rest, but this is a clearly inefficient use of the data that is available. Mathematically speaking, all unsupervised learning techniques in some sense try to 'capture' $p(x)$, the distribution from which the data was generated. This can be useful in cases such as clustering (e.g. k-means), exploratory analysis (PCA), anomaly detection (density estimation), and improving supervised classification. In the case of classification, it is easy to see through the definition of Bayes' rule that there is a relationship between $p(y|x)$ and $p(x)$:

$$p(y|x) = \frac{p(x,y)}{p(x)} = \frac{p(x|y)p(y)}{p(x)} \tag{4.1}$$

What this also means is that the estimation of $p(y|x)$ is also capturing $p(x)$, in an implicit sense. However, in the event in which we have very little labeled data, $p(y|x)$ may not be very accurate. Therefore, if we have a wealth of unlabeled data, we can try to make use of it to improve the model's understanding of $p(x)$, which will also hopefully improve $p(y|x)$.

When we have a probability distribution $p(x)$, there are two things we can do with this distribution: we can evaluate the density at a specific point x to figure out the likelihood of that input, and draw samples $x \sim p(x)$. One of the most popular techniques used in deep neural networks for unsupervised learning is the autoencoder. To recap, in the autoencoder we learn an encoding and decoding function, $f(\mathbf{x})$ and $g(\mathbf{x})$ respectively, such that we minimise (over all $\mathbf{x}$) the Euclidean distance $||\mathbf{x} - \mathbf{x}'||_2^2$, which is called the reconstruction loss. Probabilistically speaking, this can be viewed as trying to maximise $p(\mathbf{x};\theta) = \prod_{i^2} p(\mathbf{x}_i;\theta)$, assuming that each pixel $\mathbf{x}_i$ is a draw from a Gaussian distribution whose mean is $g(f(\mathbf{x}))_i$, i.e., $\mathbf{x}_i \sim \mathcal{N}(\mathbf{x}_i; \mu = g(f(\mathbf{x}))_i, \sigma^2)$. We can see however that this is a somewhat simple model as each pixel is an independent draw from a Gaussian, and furthermore, that a Gaussian distribution is unable to capture multiple modes since it is a unimodal distribution. (There are of course more sophisticated techniques such as PixelCNN (van den Oord *et al.*, 2016) where the distribution of each pixel is learned by conditioning on all the previous pixels in

the image.) Another technique however is tackling the problem through the other angle, in which we try to capture $p(x)$ through learning how to generate samples from the distribution. This is the basis behind the 'generative adversarial network' (also simply known as GAN). As of the time of writing, GANs are currently one of the most promising unsupervised learning methods within deep learning.

4.1 Introduction

In the GAN framework, we have two neural networks: a generator, and discriminator, which compete against one another. The purpose of the generator is to map (typically) from a sample of a prior distribution (which is easy to sample from) to a sample which could plausibly come from the data-generating distribution. Concretely, we wish to estimate and be able to draw samples from $q(\mathbf{x}|\mathbf{z})$, where $\mathbf{z} \sim p(\mathbf{z})$ (our prior distribution from which we can easily sample from). To ensure $q(\mathbf{x}|\mathbf{z})$ learns a convincing mapping, we introduce the second network, which is the discriminator. The discriminator, $D(\mathbf{x})$, returns the probability that $\mathbf{x}$ is a real sample (i.e., try and determine the probability $\mathbf{x}$ is from $p(\mathbf{x})$ and not $q(\mathbf{x}|\mathbf{z})$). The discriminator wants to maximize $D(\mathbf{x})$ for real $\mathbf{x}$, and minimize $D(\mathbf{x})$ for fake $\mathbf{x}$'s from the generator. The generator tries to maximize $D(\mathbf{x})$ for fake $\mathbf{x}$'s – that is, it updates its own parameters so as to fool the discriminator. Subsequently, the training of a GAN is framed as an iterative procedure where the generator tries to fool the discriminator and vice versa, until (ideally) an equilibrium is reached where the both networks are no better than outperforming the other. If we let $G(\mathbf{z})$ denote the generator (where the function $G(\mathbf{z}) = q(\mathbf{x}|z)$) we can frame the GAN loss function as the following:

$$\min_D \max_G \mathbb{E}_{\mathbf{x},\mathbf{z}}\Big[\ell(G(\mathbf{z}), 0) + \ell(D(\mathbf{x}), 1)\Big], \tag{4.2}$$

where $\{0, 1\}$ denotes fake/real, and ℓ in this case denotes the binary cross-entropy loss function:

$$\ell(\hat{y}, y) = y \log \hat{y} + (1 - y) \log (1 - \hat{y}) \tag{4.3}$$

We can instead change the maximisation of the generator into a minimisation and obtain a separate loss function for both networks:

$$\begin{aligned} &\min_D \mathbb{E}_{\mathbf{x},\mathbf{z}}\Big[\underbrace{\ell(G(\mathbf{z}), 0)}_{\text{assign prob. of 0 to fake samples}} + \underbrace{\ell(D(\mathbf{x}), 1)}_{\text{assign prob. of 1 to real samples}}\Big] \\ &\min_D \mathbb{E}_{\mathbf{x},\mathbf{z}}\Big[\underbrace{\ell(G(\mathbf{z}), 1)}_{\text{fool disc. into thinking fake is real}}\Big] \end{aligned} \tag{4.4}$$

At test time, to generate a new sample from generator, we first sample $\mathbf{z} \sim p(\mathbf{z})$ and then sample $\mathbf{z} = G(\mathbf{z})$ (which can be seen in a probabilistic manner as $\mathbf{x} \sim q(\mathbf{x}|\mathbf{z})$).

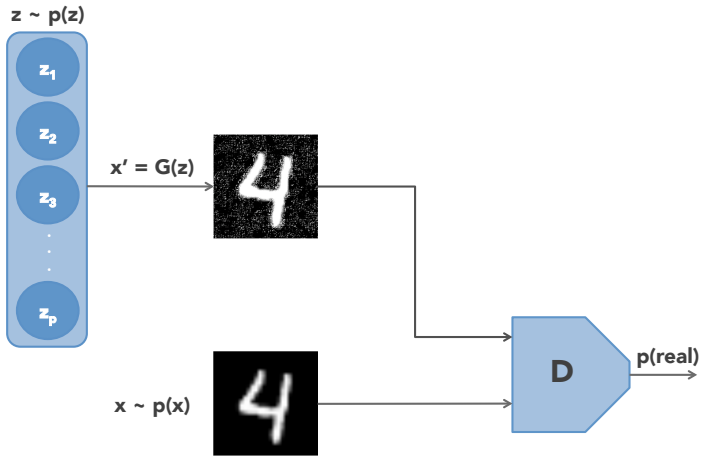


Figure 4.1 Illustration of GAN. In this, we seek a generator function G which maps samples $\mathbf{z} \sim p(\mathbf{z})$ to a generated image $\mathbf{x}'$. The discriminator D takes as input either an $\mathbf{x}'$ or $\mathbf{x} \sim p(\mathbf{x})$ and has to detect whether it is a real or generated sample. The generator tries to fool the discriminator (utilising the discriminator's gradients) so as to learn to generate images which are indistinguishable from those in the real distribution $p(\mathbf{x})$.

One might wonder what the practical purpose would be behind being learning a function to draw samples from the data distribution. The point is that in doing so, the generator ends up learning very powerful latent features – after all, these are the same features which allow it to generate believable samples. There are however two very key ideas to mention. The first is that these latent features in the generator (or discriminator) can be used to improve classification. For example, one could find a latent vector $\mathbf{z}'$ which closely represents $\mathbf{x}$ via $\min_{\mathbf{z}} ||G(\mathbf{z}) - \mathbf{x}||_2^2$. We can then feed $\mathbf{z}'$ to the generator and extract its latent outputs and use them as features. However, it turns out that bidirectional GANs have also been proposed (Dumoulin *et al.*, 2016; Donahue *et al.*, 2016) which allows one to map from $\mathbf{x} \rightarrow \mathbf{z}$ and $\mathbf{z} \rightarrow \mathbf{x}$. This also means that one can obtain the reconstruction of $\mathbf{x}$ by simply encoding to $\mathbf{z}$ and decoding back into $\mathbf{x}'$, with the benefit of also having a generative model since one can sample $\mathbf{z} \sim p(\mathbf{z})$ and also decode back into $\mathbf{x}'$.

The second key idea is to apply the adversarial loss to other techniques. For example, in the case of an autoencoder, the squared error between the original and reconstructed input is used as the error. While this is straightforward implement and understand, it does not correlate well with human perception (since imperceptible changes between two images can result in

a high reconstruction error), and reconstructions can often appear blurry for complicated natural images (since the squared error term is essentially minimising a Gaussian likelihood). Conversely, an adversarial loss has no such assumptions about the data, and several works have empirically shown that the standard reconstruction loss augmented with adversarial objectives provide better reconstructions (Larsen *et al.*, 2015; Isola *et al.*, 2016). We illustrate how one may decide to augment the autoencoder with an adversarial objective in Figure 4.2. Then, the training objectives of the autoencoder and discriminator will be:

$$\begin{aligned} &\min_{D} \mathbb{E}_{\mathbf{x}}\Big[\ell(D(\mathbf{x}), 1) + \ell(D(\mathbf{x}'), 0)\Big] \\ &\min_{g,f} \mathbb{E}_{\mathbf{x}}\Big[\lambda||\mathbf{x} - g(f(\mathbf{x}))|| + \ell(D(g(f(\mathbf{x}))), 1)\Big] \end{aligned} \tag{4.5}$$

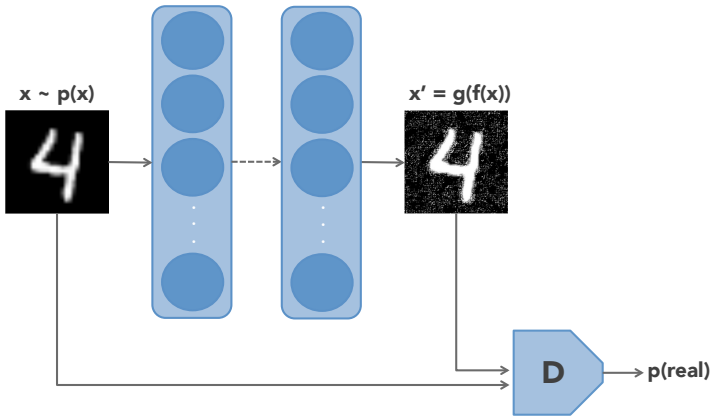


Figure 4.2 Illustration of an autoencoder augmented by discriminator + adversarial loss. The discriminator's job is to distinguish between samples from the data distribution and samples from the output of the autoencoder. Therefore, in addition to the autoencoder minimising its reconstruction term, it also has to try and fool the discriminator into thinking the reconstruction is an original input.

Recently we have seen some interesting applications of GANs, one of which in particular are image-to-image translation networks. One of these papers, image-to-image translation with conditional adversarial networks (Isola *et al.*, 2016) (which we informally refer to as 'pix2pix') explored general-purpose image translation using adversarial losses. Some example tasks included day $\rightarrow$ night, segmentation label $\rightarrow$ street scene, satellite image $\rightarrow$ street map, and so forth (as well as the inverse translations). One limitation of this work however was the requirement of paired training data, i.e., each image in one domain had to have a corresponding image in the other. To alleviate this issue, another paper was proposed

called CycleGAN (Zhu *et al.*, 2017), in which one can learn to map from one domain to another without requiring paired training data. They also managed to achieve excellent results despite the inconvenience of not having paired data (which would give a relatively stronger supervisory signal). Unlike pix2pix, in CycleGAN we actually learn a bidirectional mapping between the domains, as this is required in order to learn a meaningful mapping either direction, even if we are only interested in mapping from only one domain. This means that, depending on the application, we may end up having to do twice the work to accomplish one task (which means more memory, tuning, and computation). In particular, this work was motivated by some initial exploration in applying CycleGAN to a medical context, in which we wanted to symptomatic images (sick patients) to non-symptomatic images (healthy patients). This motivated us to pursue a computationally cheaper formulation in which we only focus on trying to map from one domain. In doing this, we found that our technique shares some conceptual similarities with GeneGAN (Zhou *et al.*, 2017), even though their work is motivated by a different task (object transfiguration). We also realised that one has to give some consideration to the relative cardinality between the two domains in order for the translations to be meaningful.

4.1.1 Our method

We first start off by explaining the ideas and mathematics behind CycleGAN, which motivates our work. Suppose we have some images belonging to one of two sets $\mathbf{x} \sim p_X(\mathbf{x})$ and $\mathbf{y} \sim p_Y(\mathbf{y})$. We wish to learn two functions (generators) $F : X \to Y$ and $G : Y \to X$ which are able to map an image from one set to the corresponding image in the other. Correspondingly, we have two discriminators $D_{\mathbf{X}}$ and $D_{\mathbf{Y}}$ which try to detect whether the image in that particular set is real or generated. In a typical fashion, we can frame the optimisation of the generator and discriminator as a min-max game, as shown:

$$\begin{aligned} \min_{D_X} \max_{G} \mathbb{E}_{\mathbf{x},\mathbf{y}} \Big[\ell(D_X(\mathbf{x}), 1) + \ell(D_X(G(\mathbf{y})), 0) \Big] \\ \min_{D_Y} \max_{F} \mathbb{E}_{\mathbf{x},\mathbf{y}} \Big[\ell(D_Y(\mathbf{y}), 1) + \ell(D_Y(F(\mathbf{x})), 0) \Big] \end{aligned} \tag{4.6}$$

where $\ell(\cdot, \cdot)$ is some classification loss. In the case of regular GAN, this is the binary cross-entropy, but Zhu *et al.* (2017) use LSGAN (Mao *et al.*, 2016) (which uses the squared error loss) for stability reasons.

To prevent the GAN from mapping any $\mathbf{x}$ to a trivial solution in $\mathbf{y}$ (and vice versa) a reconstruction penalty (referred to in the original paper was a 'cycle-consistency loss') is devised for both generators F and G. Concretely, the total objective for both generators F

and G are:

$$\min_{G,F} \mathbb{E}_{\mathbf{x},\mathbf{y}}\Big[\ell(D_X(G(\mathbf{y})),1) + \ell(D_Y(G(\mathbf{x})),1) + \lambda||\mathbf{y} - F(G(\mathbf{y}))||_1 + \lambda||\mathbf{x} - G(F(\mathbf{x}))||_1\Big] \quad (4.7)$$

We found however that the objective behind CycleGAN is one which is slightly more ambitious than the task of interest here. In CycleGAN, we wish to to find two generators F and G which can map between the two domains of interest, whereas we are only interested in modeling one direction – in our case, the mapping from non-healthy patient to healthy patient. In the CycleGAN formulation, even if one is only interested in a unidirectional mapping, the other direction has to be accounted for since the generator that maps from one domain depends on the generator in the other domain (see the cycle consistency terms in Equation 4.7), which is a somewhat cumbersome requirement (at least for training) since it requires another generator and discriminator. Furthermore, one of the cycle consistency loss term in Equation 4.7, $||\mathbf{x} - G(F(\mathbf{x}))||_1$, imposes an awkward constraint: because $G(\cdot)$ has to try and map the now-healthy patient back to a sick one, $F(\cdot)$ may preserve details of 'sickness' to try and maintain a bijection. For example, in our preliminary experiments on the zebra2horses dataset, we found it was difficult to map from zebra $\rightarrow$ horse since the stripes were not completely removed. While this is undesirable, it makes sense given the interpretation of the training objective, because if the stripes were completely removed then the opposite generator would have no idea how to map back to the original zebra.

In our technique, which we call UnitGAN (where 'unit' is short for 'unitranslational'), we restrict our focus on $F : X \rightarrow Y$, where X and are Y are the sets of images corresponding to non-healthy patients and healthy patients, respectively. Ideally however, we would like F to be a conditional function: if it receives an $\mathbf{x}$ as input, we would like this to be translated to some $\mathbf{y} \in Y$. If however it receives a $\mathbf{y}$, it should act as an identity function and simply return its input (in practice however, this would really act as an autoencoder). With this objective in mind, F should learn to detect which image belongs to what domain, and the corresponding features to detect (and remove) for any non-healthy images. Because our formulation is also adversarial, we introduce a discriminator D whose purpose is to distinguish whether an input is a real healthy image or not (i.e. is $\mathbf{y} \sim p_d(\mathbf{y})$?).

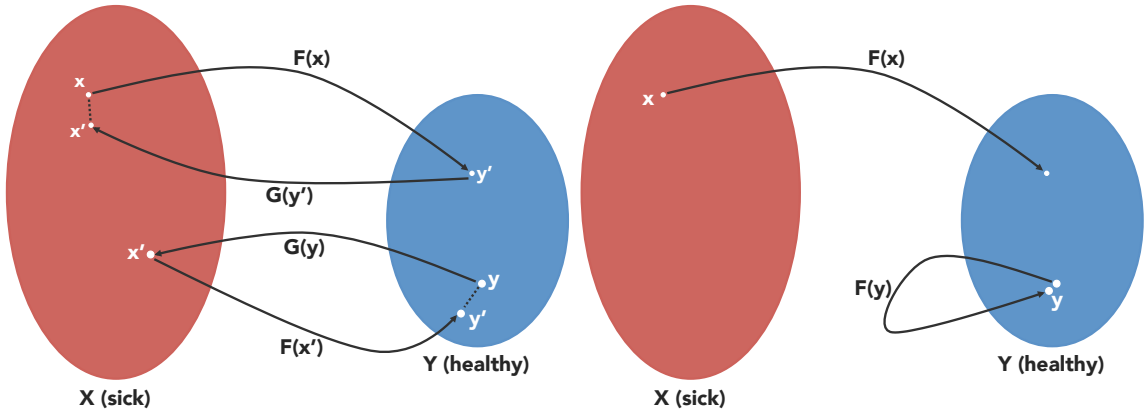
Concretely, the discriminator's objective is as follows:

$$\min_D \underbrace{\ell(D(F(\mathbf{x})),0)}_{\text{fake healthy image}} + \underbrace{\ell(D(\mathbf{y}),1)}_{\text{real healthy image}} + \underbrace{\ell(D(F(\mathbf{y})),0)}_{\text{autoencoded healthy image}} \quad (4.8)$$

And correspondingly for the generator:

$$\min_G \lambda \underbrace{||\mathbf{y} - F(\mathbf{y})||_1}_{\text{reconstruction loss}} + \underbrace{\ell(D(F(\mathbf{x})), 1)}_{\text{fake healthy image}} + \underbrace{\ell(D(F(\mathbf{y})), 1)}_{\text{autoencoded healthy image}} \tag{4.9}$$

We present an illustration summarising the differences between our formulation and CycleGAN, which is show in Figure 4.3.



(a) CycleGAN (b) UnitGAN

Figure 4.3 Illustration of CycleGAN and UnitGAN in terms of the mapping performed within and between domains. Unlike CycleGAN, in UnitGAN $F(\cdot)$ is a conditional function which can either be $Y \to Y$ or $X \to Y$.

4.1.2 Related work

It turns out the technique we proposed is almost identical to another type of GAN proposed last year by Taigman *et al.* (2016), called 'domain transfer networks' (DTN). In their work, their proposal of the generator function F is really a composition of two functions f and g (i.e., $F = f \circ g$), where f is the encoder and g is the decoding function. Apart from a domain-specific loss they introduce, they also introduce a constraint $d(f(\mathbf{x}), f(g(f(\mathbf{x}))))$ (where d is some distance function). This constraint says that if we translate $\mathbf{x}$ (by applying f then g) and then encode it again with f, it should be no different from $f(\mathbf{x})$, meaning that we want to enforce the constraint that once the source image has been translated, it cannot translate it any further. This makes sense from a model elegance point of view – if we translate an already-translated image, the result should be the input. While we also independently proposed the same loss but in pixel space, $||F(\mathbf{x}) - F(F(\mathbf{x}))||_1$, we did not have the time to explore this constraint ourselves and see its effect on translation quality. Another

difference between our works are that they evaluate their technique on much simpler datasets such as MNIST and SVHN, whereas we evaluate on significantly more complex datasets.

We note that our work bears some conceptual similarities to GeneGAN (Zhou *et al.*, 2017). In this paper the intended goal is object transfiguration in which we would like to isolate latent features corresponding to some object of interest and have the ability to add or remove them to images of interest. For instance, in their paper they were able to train a GAN to learn the latent features corresponding to 'glasses' and remove these features from faces with glasses, or add them to faces without glasses. (It is also worth noting that, just like CycleGAN, both techniques are interested in bidirectional mappings whereas we are not.) Concretely, let us override earlier notation and now denote $\mathbf{x}$ to be a sample from either the positive domain X (faces with glasses) or negative domain Y (faces without glasses). We also split F into an encoding function F_{enc} and F_{dec}. In GeneGAN, we want $F_{enc}(\mathbf{x})$ to return two encodings $[z_x^v, z_x^u]$, where z^v corresponds to the latent features we don't care about and z^u the ones corresponding to the feature of interest (e.g. the glasses). While we save specific details for their work, essentially, an adversarial objective is formed in such a way that if $x \sim p_X(\mathbf{x})$, we want $F_{enc}(\mathbf{x}) = [z_x^v, z_x^u]$, and if $x \sim p_Y(\mathbf{x})$ then we want $F_{enc}(\mathbf{x}) = [z_x^v, \epsilon]$ (we wish for $\epsilon = 0$). With this in mind, we can easily add or remove the object of interest given an image: to remove the object of interest, assuming it exists for $\mathbf{x}$, perform $F_{enc}(\mathbf{x})$, assign z_x^u to $\mathbf{0}$, and run the result through F_{dec}. If we have another image $\mathbf{y}$ which does not have the object of interest, we also encode it with F_{enc} but replace the resulting ϵ with z_x^u and decode. With their technique, they were able to achieve decent transformations based on a variety of facial attributes.

One caveat with GeneGAN is that one has to manually define the split between what in the encoding corresponds to the factor of interest (z^u) or not (z^v). For example, in the authors' code they use a simple FCN where 25% of the feature maps in the encoder output correspond to z^u. This means we unfortunately have an extra hyperparameter to tune. Conversely, in our technique the split between z^u or not z^v is implicit and F encapsulates both F_{enc} and F_{dec}.

4.2 Experiments and Results

We present experiments utilising both CycleGAN and UnitGAN in a range of problem domains. Firstly, we train both GAN formulations on the diabetic retinopathy dataset (which we made extensive use of in the previous chapter) and on the horses2zebra dataset (which was one of the datasets employed in the CycleGAN paper). While we mainly present *qualitative* results for both of these datasets, the work presented here serves as a basis to build

on further techniques which could be useful in a semi-supervised setting (which we discuss later in this chapter).

Lastly, recently we submitted a conference paper[1] which contributes a novel technique for inferring 3D keypoints of a person's face (of which 2D data is only available), which can then be used to produce a frontal view of that face. This frontal view of the face can then be mapped onto a 3D face mesh, allowing us to create different poses of a face at will. Depending on the original pose of the person however, the resulting frontalisation step can be of a very poor quality. For example, a side-on view of a face obscures the non-visible half, so the resulting frontalisation will look terrible for that half of the face because there are barely any pixels on that side of the face which can be manipulated. In order to address this issue, we proposed an application of CycleGAN which maps between two domains: frontalised faces which do not look distorted (i.e., the original poses were not extreme) and ones which do (i.e., the original poses were extreme). The distinction made between good and bad frontalisations were determined by a heuristic approach which examines and compares keypoints on the face. Once the CycleGAN has been trained, we can discard all networks except for the generator which maps from distorted face to non-distorted face and use it as a post-processing step.

4.2.1 Data

We evaluate our technique on several distinct datasets:

The horses and zebras dataset from the CycleGAN paper. This contains a pre-defined train and test split (which we use as a validation split) consisting of 1067 horses / 1334 zebras and 120 horses / 140 zebras, respectively. Images are of 256px width and height.

CelebA dataset [2]. This is an extremely large dataset consisting of roughly 200,000 celebrity faces comprising roughly 10,000 identities and 40 binary annotations per image. For this dataset, we run a face detector in order to carve out tight bounding boxes for the faces. However, because the face detector is not perfect, we end up losing roughly half of the faces in the dataset. From the remaining faces, we downsize the images to 80x80 and run them through the face frontalisation pipeline[3]. After running the heuristic detector for good/badly frontalised faces, we detected 43,114 faces as being good and 8,310 as being bad. This comprises the set of 'good' and 'bad' frontalised images for which we train CycleGAN with.

[1] Joel Moniz was the main author of the conference paper. My contribution to this was in the training of adversarial models to augment his work. This paper can be found attached to the appendix.

[2] http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

[3] This technique is not explained here since this was primarily the work of my colleague, for whom I provided assistance with.

The Kaggle diabetic retinopathy (DR) dataset[4]. This is a reasonably large dataset consisting of extremely high-resolution fundus image data. The training set consists of 17,563 pairs of images (where a pair consists of a left and right eye image corresponding to a patient). This dataset contains five levels of diabetic retinopathy: no DR (25,810 images), mild DR (2,443 images), moderate DR (5,292 images), severe DR (873 images), and proliferative DR (708 images). Because we only consider two domains in this work, we simplify the dataset by defining one set of images to be the no DR category (25,810 images) and the other to be the worst DR (708). We set aside 5% of pairs (i.e. 10%) in the training set to be part of our validation set. Images are of 256px width and height.

4.2.2 Network

For diabetic retinopathy and horses2zebra we use the 'block9' architecture identical to that used in CycleGAN. We found however that it was essential to add long skip connections between the two encoding and decoding blocks in the network as this greatly helped the quality of translation. (We suspect this is simply because in UnitGAN we only enforce a reconstruction term for $\mathbf{y}$ but never for $\mathbf{x}$, and this is corroborated by the fact that the autoencoding outputs of the GAN are much better than the translations.) For CelebA/VGG we use an FCN-like (fully convolutional network) architecture (Long *et al.*, 2015) which employs both short and long skip connections.

4.2.3 Experiments

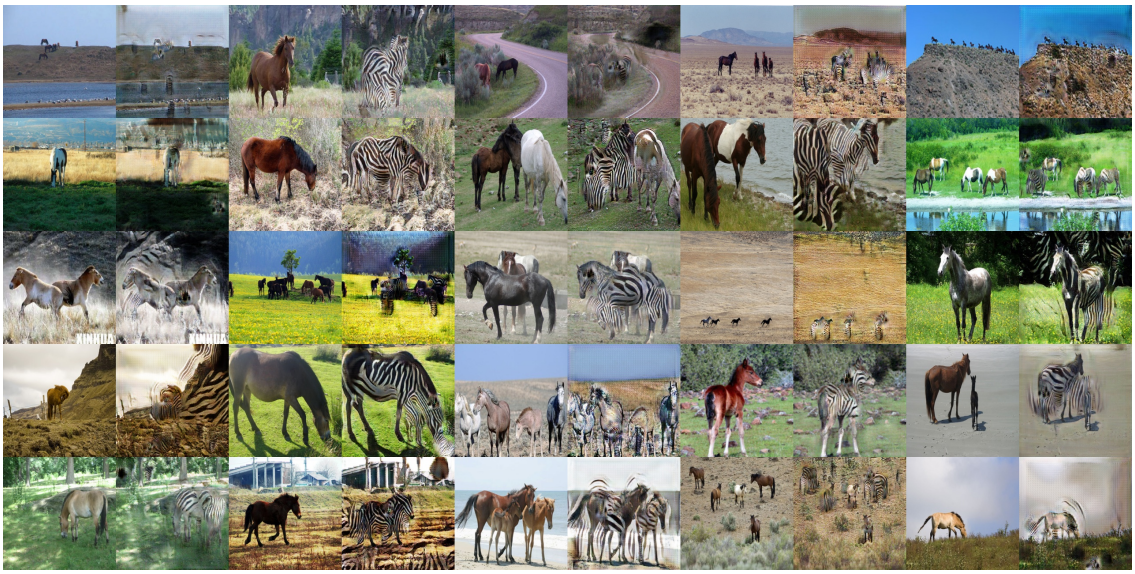
horses2zebra

In addition to it being one of the datasets used in the CycleGAN paper, we chose to train our models on this dataset as it contains significantly more variability than that of the other datasets, and therefore makes it an ideal 'stress test' for model comparison. Unfortunately, we found that our UnitGAN did not perform as well on this dataset. In particular, the translation quality was usually subpar compared to CycleGAN in the sense that we often would see weird colouring artifacts in the translations, even when we used long skip connections to pass forward information from earlier in the network. Conversely, for the autoencoding task, the reconstructions on $\mathbf{y}$ were perfect, which suggests that some sort of reconstruction loss for $\mathbf{x}$ (the image we are translating from) is important to prevent this phenomena from happening. For example, in CycleGAN a reconstruction is enforced for both $\mathbf{x}$ *and* $\mathbf{y}$. Nevertheless, we present results for both mappings in Figure 4.4, and also present the CycleGAN results in

[4]https://www.kaggle.com/c/diabetic-retinopathy-detection/

Figure 4.5.

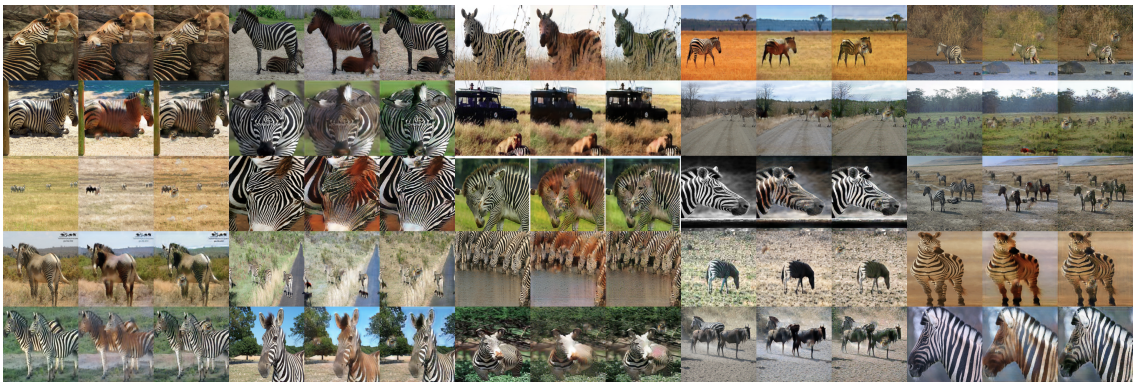
(a) Visualisations mapping from $A \rightarrow B$. Each row consists of tuples $\{A, A \rightarrow B\}$ where A denotes a zebra and B denotes a horse.



(b) Visualisations mapping from $B \rightarrow A$. Each row consists of tuples $\{B, B \rightarrow A\}$ where B denotes a horse and A denotes a zebra (i.e., the opposite direction to what was presented in Figure 4.4(a)).

Figure 4.4 Experiments on the horses/zebra image dataset with UnitGAN.

(a) Visualisations mapping from $A \to B$. Each row consists of triplets $\{A, A \to B, A \to B \to A\}$ where A denotes a horse and B denotes a zebra.



(b) Visualisations mapping from $B \to A$. Each row consists of triplets $\{B, B \to A, B \to A \to B\}$ where B denotes a zebra and A denotes a horse.

Figure 4.5 Experiments on the horses and zebras dataset with CycleGAN. In general, the translations here are more stable (in the sense that they do not produce weird colouring artifacts) than that of UnitGAN in Figure 4.4.

While the work in improving this technique is ongoing, we believe that the superiority of CycleGAN is in the loss function (even if it comes at twice the memory cost). Since one is mapping in both directions and each mapping is coupled with the other, maximal use of the data is being achieved. We also believe this is the reason why CycleGAN's translations tend to look less distorted than ours – the cycle-consistency loss forces important details to preserved and not distorted.

CelebA faces

In this work we use CycleGAN to learn how to distinguish between distorted and non-distorted faces which were frontalised by a novel 3D keypoint estimation technique. Because

there is no way to actually infer an obscured part of a face with only a single image, we can use CycleGAN to 'post-process' the frontalised face and clean up any regions for which there was very little pixel data available in the original image. The results for this can be shown in Figure 4.6, where the first, second, and third images in each row correspond to the original image, frontalised image, and CycleGAN-cleaned frontalised image, respectively.

Figure 4.6 Face repairs computed by CycleGAN. For each row we have triplets, where the first element denotes the original image, second element is the frontalised face, and third element is the post-processing performed by CycleGAN to clean up blurry or distorted parts of the face.

In order to further showcase the efficacy of our model, we also present quantitative results on the Adience dataset (which we used for ordinal prediction in Chapter 3), where we try to predict the age of a person but this time given their frontalised and cropped face. We compare this against a model where at training time each minibatch is concatenated with a version of itself where all of the faces have been treated by CycleGAN. Since this effectively acts as a form of data augmentation, we expect to see superior results to that of the baseline. For validation we do not perform any data augmentation, however. Visualisations and results are presented in Figures 4.7 and 4.8, respectively.

Perhaps what is interesting is that the CycleGAN simply does more than just fix distorted regions of a face (assuming they exist). We can see (in Figure 4.7 it even goes as far as changing the lighting of the face or even the actual expression like changing the nose of a baby (top-left corner) or converting a closed lip expression to a smile (bottom-right corner). While this certainly was not the intention, it acts as a formidable proof-of-concept to show

how generative models can be used for data augmentation.

Figure 4.7 Faces from the cropped + frontalised version of the Adience dataset and their corresponding translations to 'fixed' faces after being processed by CycleGAN.

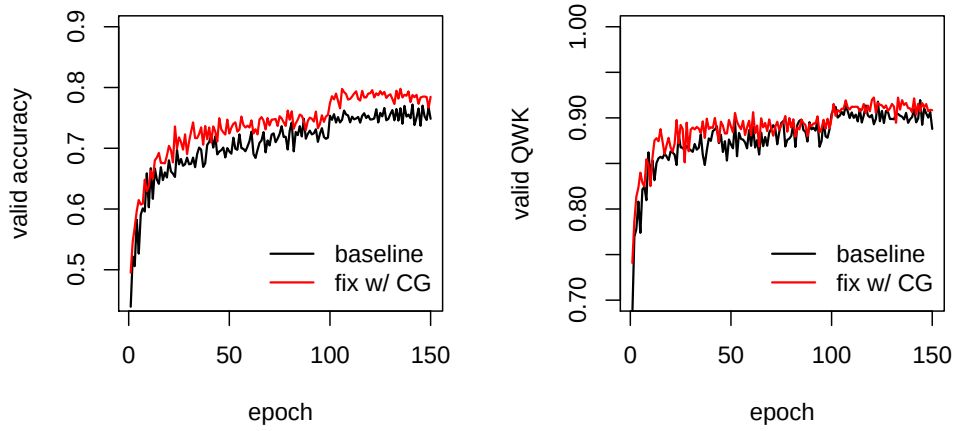


Figure 4.8 Validation accuracy and quadratic weighted kappa (QWK) on the Adience dataset. The proposed model, `fix w/ CG`, uses CycleGAN as a data augmentation technique to artificially double the size of the dataset during training by fixing any distortions in the original faces. Each experiment was run thrice with averaged curves computed.

Future work could entail using the face frontalisation work as a way to perform data augmentation to artificially increase the size of our datasets. If we have a 3D mesh of a face (with

its corresponding texture we inferred and cleaned with CycleGAN), we could construct an infinite number of different poses. The only downsides are that we would need to find ways to generate the background behind the face and the hair in order to produce a fake image which could feasibly come off as a real one. Both tasks could either be done traditionally (i.e., use computer graphics applications to add background and hair to a 3D model of a face) or through generative adversarial techniques as we have presented here. In the mean time however, we run a simple proof-of-concept where we use CycleGAN as a data augmentation technique by having it learn a forward and reverse mappings between two classes. Since this is an age estimation dataset, we are essentially learning mappings to age and de-age a face. We learn the mappings between ages $8-13$ (ordinal class 2) and ages $38-43$ (ordinal class 5). These are shown in Figure 4.9.

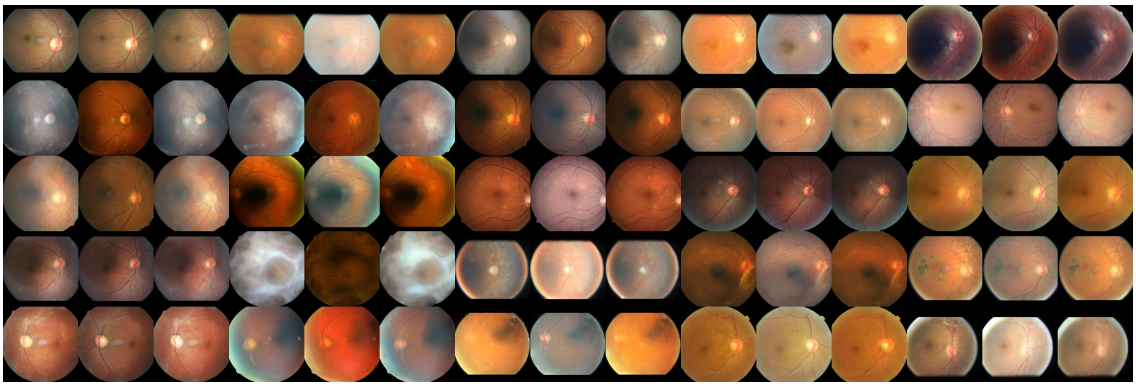
(a) Visualisations mapping from $A \rightarrow B$. Each row consists of triplets $\{A, A \rightarrow B, A \rightarrow B \rightarrow A\}$ where A denotes the age range $38-43$ and B denotes $8-13$ (i.e., old to young).

(b) Visualisations mapping from $B \rightarrow A$. Each row consists of triplets $\{B, B \rightarrow A, B \rightarrow A \rightarrow B\}$ where B denotes the age range $8-13$ and A denotes $38-43$ (i.e., young to old).

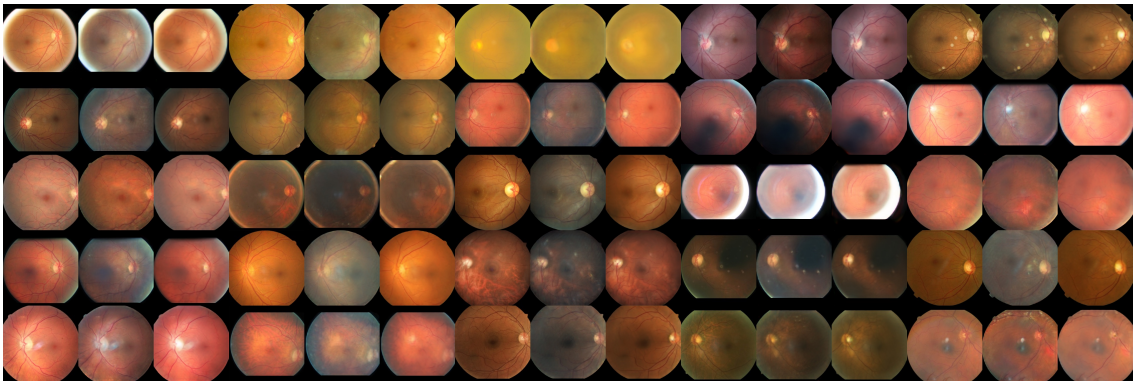
Figure 4.9 Visualisations to age and de-age faces in the Adience dataset using CycleGAN.

Diabetic retinopathy

Last but not least, we present results from both GAN formulations on the diabetic retinopathy dataset.



(a) Visualisations mapping from $A \to B$. Each row consists of triplets $\{A, A \to B, A \to B \to A\}$ where A denotes a symptomatic retina and B denotes a normal one.



(b) Visualisations mapping from $B \to A$. Each row consists of triplets $\{B, B \to A, B \to A \to B\}$ where B denotes a normal retina and B denotes a symptomatic one (i.e., the opposite direction to what was presented in Figure 4.10(a)).

Figure 4.10 Experiments on the raw diabetic retinopathy image dataset with CycleGAN.

In Figure 4.10 we show results from the CycleGAN baseline on the diabetic dataset, where Figure 4.10(a) shows transformations in the $A \to B$ direction, i.e., non-healthy to healthy, and Figure 4.10(b) shows transformations in the opposite direction). We can see that the GAN is able to localise and remove symptoms while being somewhat robust to different factors of variation such as the model of camera, whether the image is left/right eye, inverted, and so forth. At the same time, we can sometimes see that the GAN will completely change the overall colour of the retinal image, as can be seen in the first triplets of the second and third rows of Figure 4.10(a). However, given the heterogeneity of the dataset and the fact that

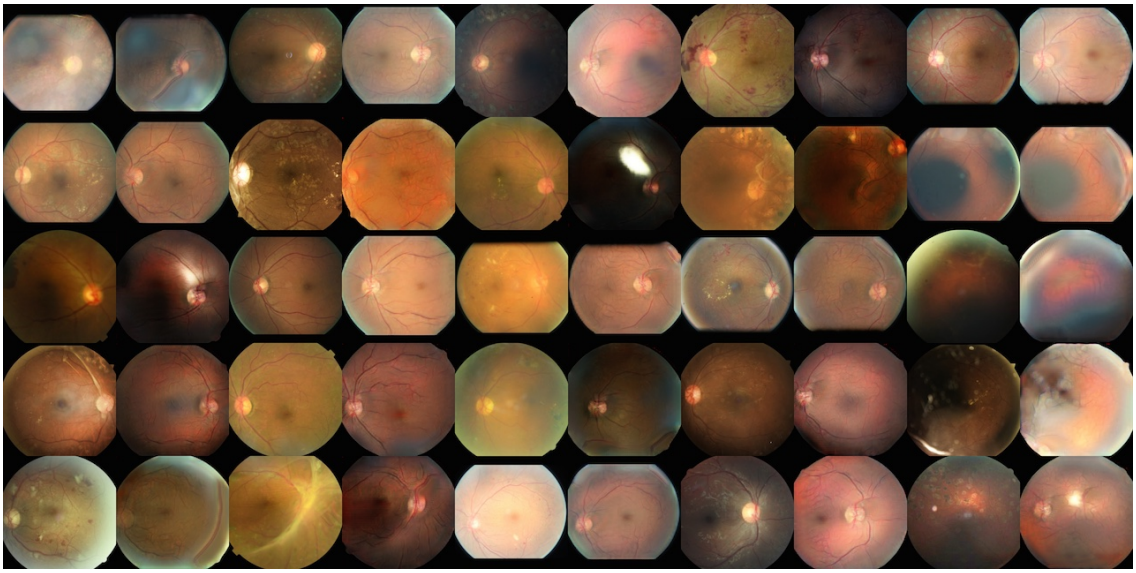


Figure 4.11 Experiments on the raw diabetic retinopathy dataset with UnitGAN. Each row consists of tuples $\{A, A \rightarrow B\}$ where A denotes a symptomatic (non-healthy) image and B denotes a healthy one.

many of the images were taken under different kinds of cameras and lighting conditions, this is be expected. While it is trivial to standardise the dataset so that each retinal image looks roughly the same in terms of colour and lighting, we wanted to train generative models which required little preprocessing of the input data.

Because the dataset is quite large, it is not practical to manually evaluate the quality of image transformations performed by the network, and we would like to perform this task automatically. Similar to the idea behind the proposed Inception score (which was proposed as a way to determine the realism of samples generated by a GAN) Salimans *et al.* (2016), we employ a pre-trained classifier which predicts the level of diabetic retinopathy and evaluate this model on images before and after they have been fed into the network. If, for example, the GAN has been trained to transform symptomatic images to non-symptomatic, we expect to see changes in the probability distribution of those transformed images. Therefore, if we let $F(\mathbf{x}_i) = \mathbf{y}'_i$ denote a transformed sick image (either from CycleGAN or our GAN), we can compute statistics such as the mean and variance of $p(y = 0|\mathbf{y}')$ across both the training and validation set (where $y = 0$ denotes healthy). The result of this can be seen in Figure 4.12.

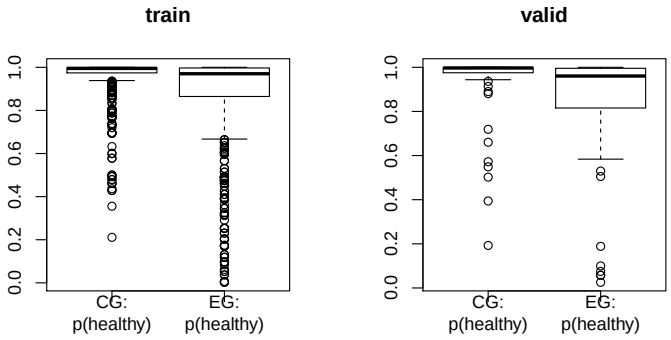


Figure 4.12 p(healthy) on transformed sick images $\mathbf{x}$ on both training and validation sets using a pre-trained diabetic retinopathy classifier. (CG = CycleGAN, EG = UnitGAN)

Unfortunately our proposed method suffers from a higher variance on both the training and validation sets, which means that CycleGAN tends to be more consistent in producing translations which the classifier is confident in. We believe this is because no reconstruction loss is being applied over $\mathbf{x}$ – which is the input we are translating – whereas in CycleGAN a reconstruction loss is imposed due to the cycle-consistency loss between $\mathbf{x}$ and $G(F(\mathbf{x}))$ (but this of course requires one to learn a G to map in the opposite direction). We believe this improves translation as the backwards mapping forces the forward mapping to be conservative in what details are added since any extraneous details will have to then be removed in the backwards mapping. For example, we found that on horses/zebras UnitGAN tends to be less stable and distort the background more often.

4.2.4 Discussion

Research in image-to-image translation becomes more interesting if think about the relative cardinalities of the domains from which we are mapping. For example, in the context of medical imaging, if we consider the domain of sick patients X and the domain of healthy patients Y, we can imagine the former being a much larger domain. In other words, there are relatively fewer ways to make a sick patient healthy than there is to make a healthy patient sick. When we consider the loss terms in the original CycleGAN loss (see Equation 4.7) we can see a potential issue. Let us consider the cycle consistency loss corresponding to mapping from sick $\rightarrow$ healthy $\rightarrow$ sick, which is $||\mathbf{x} - G(F(\mathbf{x}))||_1$. For this term, we are saying that when we map from sick to healthy, we must also find a way to go back to the same image, which is illustrated in Figure 4.13. This is a somewhat odd constraint, and for

two reasons. The first is that because of the constraint, the generator F may try to 'cheat' and preserve details in the healthy image $\mathbf{y}' = F(\mathbf{x})$ so as to find an easy way to map back to $\mathbf{x}'$ in the original domain (this can be seen in zebra $\rightarrow$ horses translation, in which the generator will leave some subtle stripes on the zebrified horse in order to be able to map back to the original zebra). The second reason is that there are many different ways to map from a healthy to sick patient, and there should be no penalty in transforming $\mathbf{y}'$ back into either a different type of sickness, or the same type of sickness but located somewhere else in the image. This means that if the domain we're mapping to is of a higher cardinality, we should be doing two things: 1) injecting noise in the mapping (to encourage exploration of the many possible outcomes); and 2) in the case of CycleGAN loosen the cycle-consistency loss corresponding to sick $\rightarrow$ healthy $\rightarrow$ sick (or do away with it some how).

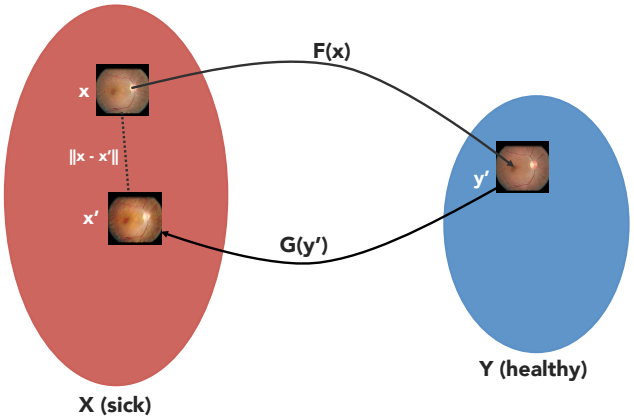


Figure 4.13 Illustration showing the CycleGAN mapping from sick $\rightarrow$ healthy $\rightarrow$ sick.

One particularly interesting application of mapping from healthy $\rightarrow$ sick is that it gives us the ability to perform a very sophisticated form of data augmentation which goes beyond what is usually done in the training of deep networks.

The other cycle consistency loss $||\mathbf{y} - F(G(\mathbf{y}))||_1$ makes slightly more sense since there is less variability within the same healthy patient. In UnitGAN this is the only reconstruction loss we enforce – for the mapping from sick $\rightarrow$ healthy there is no reconstruction loss. This can be illustrated in Figure 4.14.

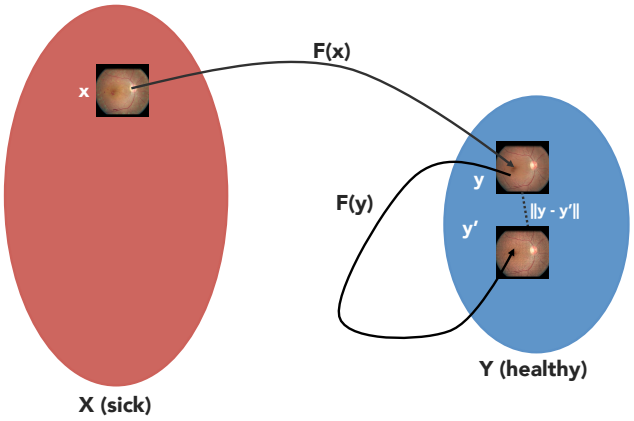


Figure 4.14 Illustration showing the UnitGAN mapping from sick → healthy → sick.

We note that if we were interested in mapping in the opposite direction – that is, from healthy → sick – CycleGAN would make more sense. This is because in this direction the cycle-consistency loss $||\mathbf{y} - F(G(\mathbf{y}))||_1$ makes more sense due to there being less variability within the same healthy patient compared to the same sick patient. Conversely, in our GAN formulation we would end up enforcing the reconstruction loss between the sick image and its reconstruction, which, as we mentioned earlier in this discussion, is an odd constraint as it prevents the function from making other semantically meaningful transformations, such as maybe changing the type of sickness or shifting the already-present symptoms.

Earlier we mentioned that it is possible that the cycle-consistency loss could be hindering translation quality, and motivated the example of zebra to horse translation, where the network may try to leave stripes on the 'zebraified horse' so that it is able to map back to the same horse when it performs the cycle (note that a similar analogy can be made for the diabetic retinopathy dataset, where symptoms may not end up being completely removed). Interestingly enough, it would be trivial to convert CycleGAN to UnitGAN through removal and addition of certain loss terms, which allows for the possibility for a two-stage training procedure where in the first stage we train CycleGAN as usual, but in the second stage we add and remove certain loss terms to convert it to UnitGAN, which can be seen as a fine-tuning step since translation quality would no longer be hindered by the cycle-consistency term. We are currently exploring this possibility.

4.3 Conclusion

In this chapter we introduced generative adversarial networks, explained their utility, and presented adversarial image-to-image translation experiments across a range of datasets for

both CycleGAN and a GAN that we proposed. We also presented some results to showcase the usefulness in leveraging image-to-image translation for data augmentation, which is a crucial ingredient to augment classification performance. In terms of future work, we would like to:

Explore the effect of a deeper architectures for both our formulation and CycleGAN. In the original CycleGAN paper, the authors use a somewhat simple ResNet architecture consisting of a preprocessing block, two encoder blocks (each downsample their inputs by 2), a variable number of resblocks (which are seen as the transformative blocks of the network) followed by two decoder blocks (which upsample their inputs by 2 in the form of deconvolution) and the final convolution which outputs the translation. Because we only ever downsample at most by 2^2, this means that for a 256×256 input the lowest resolution we operate in is 64×64, which corresponds to a 4×4 receptive field for each pixel in the transformation blocks. A deeper architecture – one which operates on a lower resolution – should in theory allow for more global transformations which could be beneficial. In fact, we tried this precise kind of architecture for the face frontalisation work we presented earlier, and we would like to adapt this architecture for both the diabetic and horses/zebra datasets.

For mappings where the codomain is larger in cardinality, inject noise in the mapping so that the network is able to explore different possibilities. As we mentioned earlier, a limitation of both GAN techniques are that they do not allow the network to explore different options for when the codomain is of a larger cardinality than the domain (e.g., there are many different ways to generate a sick patient from a healthy one). Without conditioning on some form of noise, we would not thoroughly explore the codomain and this could limit the kinds of synthetic examples we produce.

Explore ways to improve the translation quality. Earlier in the chapter we discussed how the cycle-consistency loss in CycleGAN was essential for training stability but could potentially hinder translation quality. One technique we would like to explore is to combine the loss functions of both CycleGAN and UnitGAN and experiment with a two-stage training procedure where the first stage consists of training CycleGAN as usual, and the second stage gradually anneals the cycle-consistency constraints.

CHAPTER 5 GENERAL DISCUSSION

Having gained experience in generative adversarial networks and ordinal classification, an obvious idea would be to combine the two and use the former to improve classification performance in the latter. For both CycleGAN and UnitGAN we looked at image-to-image translation in the context of mapping between two domains. This was the case even for our diabetic retinopathy dataset, where we simplified matters by binarising the dataset so that we only concerned ourselves with the healthy class (no DR) and the most proliferative form of retinopathy. In future work we would like to explore a version of CycleGAN in which we are still mapping between two domains, but where the 'sick' domain now encompasses multiple levels of sickness. In the case of diabetic retinopathy, these would be the four stages: mild DR, moderate DR, severe DR, and proliferative DR. This could potentially be achieved by simply conditioning both generators on an ordinal label denoting the level of sickness we would like to map to (or map from) and also having the discriminators in the network try to detect if the generated image is in the correct ordinal category. We have illustrated how this could be done in Figure 5.1. In this figure, the domain X now encompasses different levels of sickness $\{1, \ldots, K\}$, and $F(\cdot)$ will map any level of sickness to the healthy class. However, for the backwards mapping we condition the generator G on k, since there are different levels of sickness one could map to from healthy.

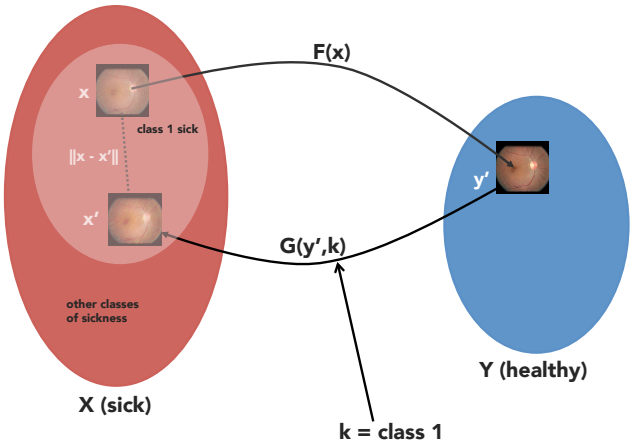


Figure 5.1 Illustration of an ordinal version of CycleGAN. X now denotes any level of sickness. While F (the mapping from sick to healthy) remains unchanged, we can now condition the opposite mapping G on an (ordinal) label k, which denotes the level of sickness we would like to map to.

Alternatively, we could propose F to be the function that learns a mapping from class k to class $k-1$, and use G to map backwards from $k-1$ to k. What this means is that one could in theory apply F repeatedly to $\mathbf{x}$ some number of times to map it all the way down to the desired class, for example $F(\mathbf{x})$ to go down to $k-1$, $F(F(\mathbf{x}))$ to go down to $k-2$, and so forth. Likewise, we could use the backwards mapping G to map from k to $k+1$ and so forth. This would also be very computationally efficient as one would not need to train roughly K^2 CycleGAN models (i.e., all the potential pairs of different classes one could map up/down from) to perform this task.

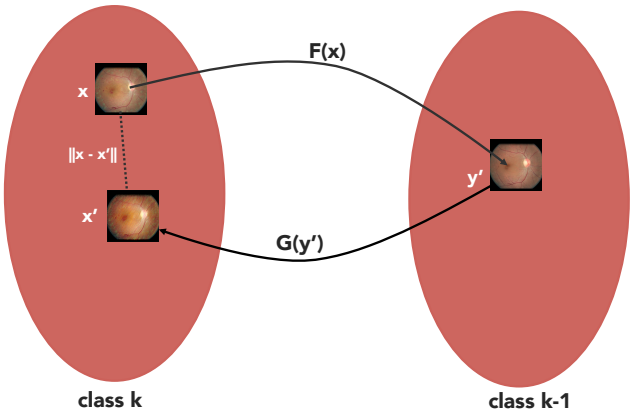


Figure 5.2 Another formulation for an ordinal version of CycleGAN. Unlike in Figure 5.1, the generator F's task is to map from class k to class $k-1$, and the opposite mapping is performed by G. This means that in order to map down to class $k-j$, we apply the function F j times.

One caveat however is that unlike mapping to the healthy class, we would still expect there to be some stochasticity which is required even from mappings that go down from k to $k-1$, since there still may be a multitude of different ways to make a 'very sick' person 'slightly sick'.

While we have not gone into the mathematical details that would go into the optimisation of either of the two models, the second proposed model seems a bit trickier to implement. This is because one may have to end up making the resulting math quite complicated by implementing 'higher-order' interactions between any two classes. For example, if we wanted to enforce cycle-consistency between class k and 0 (the healthy class) we would have to add terms $||\mathbf{x} - F(F(\ldots(\mathbf{x})))||$, whereas in the first model we learn a direct mapping to this class.

CHAPTER 6 CONCLUSION AND RECOMMENDATIONS

In this work we presented two interesting and important research topics in the realm of deep medical imaging: ordinal classification and generative adversarial networks. We first introduced ordinal classification, a problem in which the classes to be predicted are discrete but follow some ordering relationship – in a medical context, an example of this would be the various stages of cancer. What separates ordinal problems from discrete ones are that certain misclassifications carry more weight than others, which means we must try to avoid making predictions which could be risky. In our work, we explored the problem of non-unimodality in ordinal probability distributions and explained why they were not semantically meaningful. We addressed this issue by constraining the outputs of the deep neural network to predict parameters of a discrete unimodal distribution such as a binomial or Poisson, which in turn was used to generate a unimodal probability distribution over the classes. Empirically, we achieved competitive results and provided evidence to show that the enforcement of unimodality can also act as a regulariser over the probability distribution, especially in a low data regime. Because obtaining labeled data is still quite expensive in many different applications (including medical imaging), regularisation in low data regimes are still extremely useful to consider.

In the second work, we introduced generative adversarial networks (GANs) and explained their importance in the context of unsupervised and semi-supervised learning, which has garnered strong interest due to the impracticality of obtaining a large amount of labeled data for many different applications. In particular, we explored adversarial image-to-image translation using two different techniques in the literature, one being CycleGAN and the other being domain translation networks (which we independently re-invented with our formulation, UnitGAN). We utilised these techniques in order to be able to transform retinal scans of unhealthy patients (those with diabetic retinopathy) to their healthy counterparts. In doing so, we learned a model which could successfully localise and remove parts of the image which were symptomatic, despite the fact that we had no rich labels to identify these symptoms, such as bounding boxes or segmentations. This could for example be used in a semi-supervised framework to augment a network with relies on scarcely available 'rich' labels. To reinforce this notion, we presented quantitative results showcasing CycleGAN's utility in generating synthetic examples to improve classification performance in face age estimation.

In addition to comparing and contrasting some differences between CycleGAN and UnitGAN, we proposed some potential improvements to both models for which we would like to

execute on in the future, such as exploring the use of image-to-image translation for data augmentation through the addition of a noise conditioning component. We also suggested a topic which sits at the intersection of image-to-image translation and ordinal classification, which is conditioning an image translation on ordinal labels so that we can learn a generative model which (for example) learns to map a healthy patient to different levels of sickness.

REFERENCES

Beckham, Christopher and Pal, Christopher (2016). A simple squared-error reformulation for ordinal classification. *arXiv preprint arXiv:1612.00775*.

Beckham, Christopher and Pal, Christopher (2017). Unimodal probability distributions for deep ordinal classification. *International Conference on Machine Learning*. 411–419.

Bourlard, Hervé and Kamp, Yves (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, *59*(4), 291–294.

Jianlin Cheng (2007). A neural network approach to ordinal regression. *CoRR*, *abs/0704.1028*.

Chollet, François and others (2015). Keras. `https://github.com/fchollet/keras`.

Cohen, Jacob (1968). Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological bulletin*, *70*(4), 213.

da Costa, Joaquim F Pinto and Alonso, Hugo and Cardoso, Jaime S (2008). The unimodal model for the classification of ordinal data. *Neural Networks*, *21*(1), 78–91.

Sander Dieleman and Jan Schlüter and Colin Raffel and Eben Olson and Søren Kaae Sønderby and Daniel Nouri and et al (2015). Lasagne: First release.

Donahue, Jeff and Krähenbühl, Philipp and Darrell, Trevor (2016). Adversarial feature learning. *arXiv preprint arXiv:1605.09782*.

Dumoulin, Vincent and Belghazi, Ishmael and Poole, Ben and Lamb, Alex and Arjovsky, Martin and Mastropietro, Olivier and Courville, Aaron (2016). Adversarially learned inference. *arXiv preprint arXiv:1606.00704*.

Dumoulin, Vincent and Visin, Francesco (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Eidinger, Eran and Enbar, Roee and Hassner, Tal (2014). Age and gender estimation of unfiltered faces. *IEEE Transactions on Information Forensics and Security*, *9*(12), 2170–2179.

Frank, Eibe and Hall, Mark (2001). A simple approach to ordinal classification. *European Conference on Machine Learning*. Springer, 145–156.

Gal, Yarin and Ghahramani, Zoubin (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *international conference on machine learning*. 1050–1059.

Gentry, Amanda Elswick and Jackson-Cook, Colleen K and Lyon, Debra E and Archer, Kellie J (2015). Penalized ordinal regression methods for predicting stage of cancer in high-dimensional covariate spaces. *Cancer informatics*, *14*(Suppl 2), 201.

Ben Graham (2015). Kaggle diabetic retinopathy detection competition report.

Gutiérrez, Pedro Antonio and Perez-Ortiz, Maria and Sanchez-Monedero, Javier and Fernández-Navarro, Francisco and Hervas-Martinez, Cesar (2016). Ordinal regression methods: survey and experimental study. *IEEE Transactions on Knowledge and Data Engineering*, *28*(1), 127–146.

Gutiérrez, Pedro Antonio and Tiňo, Peter and Hervás-Martínez, César (2014). Ordinal regression neural networks based on concentric hyperspheres. *Neural Netw.*, *59*, 51–60.

Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun (2015). Deep residual learning for image recognition. *CoRR*, *abs/1512.03385*.

Le Hou and Chen-Ping Yu and Dimitris Samaras (2016). Squared earth mover's distance-based loss for training deep neural networks. *CoRR*, *abs/1611.05916*.

Sergey Ioffe and Christian Szegedy (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, *abs/1502.03167*.

Phillip Isola and Jun-Yan Zhu and Tinghui Zhou and Alexei A. Efros (2016). Image-to-image translation with conditional adversarial networks. *CoRR*, *abs/1611.07004*.

Khan, Mohammad E and Mohamed, Shakir and Marlin, Benjamin M and Murphy, Kevin P (2012). A stick-breaking likelihood for categorical data analysis with latent gaussian models. *International conference on Artificial Intelligence and Statistics*. 610–618.

Diederik P. Kingma and Jimmy Ba (2014). Adam: A method for stochastic optimization. *CoRR*, *abs/1412.6980*.

Kingma, Diederik P and Welling, Max (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Koren, Yehuda and Sill, Joe (2011). Ordrec: an ordinal model for predicting personalized item rating distributions. *Proceedings of the fifth ACM conference on Recommender systems*. ACM, 117–124.

Kotsiantis, Sotiris B and Pintelas, Panagiotis E (2004). A cost sensitive technique for ordinal classification problems. *Hellenic Conference on Artificial Intelligence*. Springer, 220–229.

Krizhevsky, Alex and Sutskever, Ilya and Hinton, Geoffrey E (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*. 1097–1105.

Anders Boesen Lindbo Larsen and Søren Kaae Sønderby and Ole Winther (2015). Autoencoding beyond pixels using a learned similarity metric. *CoRR*, *abs/1512.09300*.

LeCun, Yann A and Bottou, Léon and Orr, Genevieve B and Müller, Klaus-Robert (2012). Efficient backprop. *Neural networks: Tricks of the trade*, Springer. 9–48.

Long, Jonathan and Shelhamer, Evan and Darrell, Trevor (2015). Fully convolutional networks for semantic segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3431–3440.

Xudong Mao and Qing Li and Haoran Xie and Raymond Y. K. Lau and Zhen Wang (2016). Multi-class generative adversarial networks with the L2 loss function. *CoRR*, *abs/1611.04076*.

McCullagh, Peter (1980). Regression models for ordinal data. *Journal of the royal statistical society. Series B (Methodological)*, 109–142.

Niu, Zhenxing and Zhou, Mo and Wang, Le and Gao, Xinbo and Hua, Gang (2016). Ordinal regression with multiple output cnn for age estimation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4920–4928.

Tim Salimans and Ian J. Goodfellow and Wojciech Zaremba and Vicki Cheung and Alec Radford and Xi Chen (2016). Improved techniques for training gans. *CoRR*, *abs/1606.03498*.

Schapire, Robert E and Stone, Peter and McAllester, David and Littman, Michael L and Csirik, János A (2002). Modeling auction price uncertainty using boosting-based conditional density estimation. *ICML*. 546–553.

Simonyan, Karen and Zisserman, Andrew (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Srivastava, Nitish and Hinton, Geoffrey E and Krizhevsky, Alex and Sutskever, Ilya and Salakhutdinov, Ruslan (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, *15*(1), 1929–1958.

Yaniv Taigman and Adam Polyak and Lior Wolf (2016). Unsupervised cross-domain image generation. *CoRR*, *abs/1611.02200*.

Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, *abs/1605.02688*.

Aäron van den Oord and Nal Kalchbrenner and Oriol Vinyals and Lasse Espeholt and Alex Graves and Koray Kavukcuoglu (2016). Conditional image generation with pixelcnn decoders. *CoRR*, *abs/1606.05328*.

Vincent, Pascal and Larochelle, Hugo and Bengio, Yoshua and Manzagol, Pierre-Antoine (2008). Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th international conference on Machine learning*. ACM, 1096–1103.

Zeiler, Matthew D and Fergus, Rob (2014). Visualizing and understanding convolutional networks. *European conference on computer vision*. Springer, 818–833.

Shuchang Zhou and Taihong Xiao and Yi Yang and Dieqiao Feng and Qinyao He and Weiran He (2017). Genegan: Learning object transfiguration and attribute subspace from unpaired data. *CoRR*, *abs/1705.04932*.

Jun-Yan Zhu and Taesung Park and Phillip Isola and Alexei A. Efros (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, *abs/1703.10593*.

ANNEXE A Ordinal classification

Due to page limit requirements in our published ICML paper on ordinal classification, we take the time to elaborate on certain parts of our work in this section. In particular, we provide an expanded literature review section and explain our evaluation metric (the quadratic weighted kappa).

Related work (extended)

Our work is inspired by the recent work of Hou *et al.* (2016), who shed light on the issues associated with different probability distributions having the same cross-entropy loss for ordinal problems. In their work, they alleviate this issue by minimising the 'Earth mover's distance' (also called the Wasserstein distance), in which the distance is the minimum cost needed to transform one probability distribution to another.[1] Because this metric takes into account the distances between classes – moving probability mass to a far-away class incurs a large cost – the metric is appropriate to minimise for an ordinal problem. It turns out that in the case of an ordinal problem, the Earth mover's distance reduces down to Mallow's distance:

$$\text{emd}(\hat{\mathbf{y}}, \mathbf{y}) = \left(\frac{1}{K}\right)^{\frac{1}{l}} ||\text{cmf}(\hat{\mathbf{y}}) - \text{cmf}(\mathbf{y})||_l, \tag{A.1}$$

where $\text{cmf}(\cdot)$ denotes the cumulative mass function for some probability distribution, $\mathbf{y}$ denotes the ground truth (one-hot encoded), $\hat{\mathbf{y}} = p(\mathbf{y}|\mathbf{x})$ the corresponding predicted probability distribution, and K the number of classes. The authors evaluate the EMD loss on two age estimation and one aesthetic estimation dataset and obtain state-of-the-art results. However, the authors do not show comparisons between the probability distributions learned between EMD and cross-entropy. Furthermore, the loss function is simply a soft constraint on enforcing unimodality, rather than a hard constraint like having the neural network output parameterise a unimodal probability distribution (which is what we propose). Rather than build upon this work, we instead take an alternate angle to addressing the unimodality issue.

A very important note we need to make is that unimodality had already been explored for ordinal neural networks in da Costa *et al.* (2008).[2] They explored the use of the binomial and

[1] The Wasserstein distance is actually one of many different distance functions defined on probability distributions. One can prove that minimising the KL divergence (another metric) between the true distribution and estimate is equivalent to minmising cross-entropy.

[2] Unfortunately, a fact we had only found out after our work was accepted for publication.

Poisson distributions and a non-parametric way of enforcing unimodal probability distributions (the latter of which we do not explore in this work). One key difference between their work and ours here is that we evaluate these unimodal distributions in the context of deep learning, where the datasets are generally much larger and have more variability. Some other differences include introducing the tau term to control the variance of the distribution being used, and also evaluating the techniques in conjunction with the Earth mover's distance loss (Hou *et al.*, 2016).

As we mentioned earlier, there is a somewhat ample literature surrounding ordinal classification. Recently, Gutiérrez *et al.* (2016) proposed a taxonomic tree to classify the various kinds of ordinal classification techniques, some of these including: naive techniques; ordinal binary decompositions; and threshold-based approaches. Naive techniques include treating the problem like a regression or employing cost-sensitive classification (Kotsiantis and Pintelas, 2004); ordinal binary decompositions involve techniques like training several binary classifiers (Frank and Hall, 2001) or training a single classifier that can handle multiple outputs (say, a neural network) (Cheng, 2007); and threshold-based approaches such as the proportional odds model (McCullagh, 1980) and ensembles, to name a few. While the proposed taxonomy is useful in the sense that it helps draw comparisons between the different categories, it is not necessarily the case that a technique exclusively belongs to a category. For example, since we are dealing with neural networks for this thesis, we limit our focus to binary decomposition techniques as these include models which can predict multiple outputs (such as a neural networks). However, one could easily generalise a threshold-based approach to a neural network, and we will actually talk about threshold-based approaches later in this chapter.

For neural networks, Cheng (2007) proposed the use of binary cross-entropy or squared error on an 'ordinal encoding scheme' rather than the one-hot encoding which is commonly used in discrete classification (this technique was also employed recently by Niu *et al.* (2016) in CNNs). For example, if we have K classes, then we have labels of length $K-1$, where the first class is $[0, \ldots, 0]$, second class is $[1, \ldots, 0]$, third class is $[1, 1, \ldots, 0]$ up to $[1, \ldots, 1]$.[3] With this formulation, we can think of the i'th output unit as computing the cumulative probability $p(y > i|\mathbf{x})$, where $i \in \{0, \ldots, K-2\}$ (note that we do not need to model $p(y > K-1)$ since we index starting from zero). If we have cumulative probabilities then it is trivial to define

[3]We will always index probabilities from zero for the remainder of this thesis.

the corresponding discrete probabilities as the following:

$$p(y = i|\mathbf{x}) = \begin{cases} 1 - p(y > i|\mathbf{x}) \text{ if } i = 0 \\ p(y > i - 1|\mathbf{x}) - p(y > i|\mathbf{x}) \text{ otherwise} \\ p(y > K - 2|\mathbf{x}) \ i = K - 1 \end{cases} \tag{A.2}$$

It turns out that Frank and Hall (2001) proposed this scheme much earlier but in a more general sense by introducing the notion that one can use multiple models (not necessarily neural networks) to model each cumulative probability and estimate the discrete probabilities as shown in Equation A.2. In that case, the method proposed by Cheng (2007) is actually a special case in which a single neural network with multiple outputs is used instead, which is statistically and computationally more efficient. This technique however suffers from the issue that the cumulative probabilities $p(y > 0|\mathbf{x}), \ldots, p(y > K - 2|\mathbf{x})$ are not guaranteed to be monotonically decreasing, which means that if we compute the discrete probabilities $p(y = 0|\mathbf{x}), \ldots, p(y = K - 1|\mathbf{x})$ these are not guaranteed to be strictly positive because each cumulative probability of a class is independent of the other probabilities. In fact, because of this, the author does not use Equation A.2 for neither the training nor the prediction. For training, the binary cross-entropy loss is used over each output probability, i.e:

$$\ell(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{K} \mathbf{y}_i \cdot p(\mathbf{y}_i|\mathbf{x}) + (1 - \mathbf{y}_i) \cdot (1 - p(\mathbf{y}_i|\mathbf{x})) \tag{A.3}$$

At prediction time we simply compute the one-hot vector $\mathbf{1}_{\{\hat{y}_i > 0.5\}}$ by thresholding the computed probabilities $\hat{\mathbf{y}} = p(\mathbf{y}|\mathbf{x})$. However, due to the monotonicity issue explained earlier, we may obtain nonsense outputs such as $[1, 1, 0, 1]$ or $[0, 0, 1, 0]$. To alleviate this, Cheng proposes a simple heuristic in which the location of the last 1 in the vector denotes the predicted class (if there is no 1, then we predict the first class). One may wonder at this point why we are going through the hassle of defining cumulative probabilities especially when we are actually interested in discrete probabilities, since we want to do classification. We will hopefully make this detail more clear in the next upcoming paragraphs.

For now, as we mentioned earlier, one issue with the approach proposed by Cheng (2007); Frank and Hall (2001) is that the estimated cumulative probabilities are not necessarily monotonic. Perhaps surprisingly, the formulation denoted by Cheng is very similar to that of the proportional odds model (POM), which does not suffer from this issue since it learns monotonically increasing bias terms in the calculation of its cumulative probabilities. In the proportional odds model (POM) we want to estimate the cumulative probabilities $p(y \leq$

$0|\mathbf{x}), \ldots, p(y \leq K-1|\mathbf{x})$.[4] If we let $f(\mathbf{x}) = \theta^T\mathbf{x} \in \mathbb{R}$ denote the output of $\mathbf{x}$, then $p(y \leq j|\mathbf{x}) = sigm(f(\mathbf{x};\theta)+b_j)$, where b_j denotes a learned bias for the j'th class and $b_0 < b_1 < \cdots < b_{K-1}$. Then, $p(y = j|\mathbf{x})$ is simply:

$$p(y = j|\mathbf{x}) = \begin{cases} p(y \leq j|\mathbf{x}) - p(y \leq j-1|\mathbf{x}) & \text{if } j \in \{1, \ldots, K-2\} \\ p(y \leq j|\mathbf{x}) & \text{if } j = 0 \\ 1 - \sum_{i=1}^{K-2} p(y = i|\mathbf{x}) & \text{if } j = K-1 \end{cases} \tag{A.4}$$

We can easily generalise techniques like this to neural networks by simply allowing $f(\mathbf{x})$ to be the output of a neural network. In essence, for POM, the cumulative probability of a particular class $p(y \leq i|\mathbf{x})$ is $sigm(\mathbf{h}^{(l-1)}\mathbf{W} + \mathbf{b})$, where $\mathbf{W} \in \mathbb{R}^{p\times 1}$, $\mathbf{h}^{(L-1)}$ is the output of the layer before the classification layer, $\mathbf{b}$ is a monotonically increasing bias vector, and the discrete probabilities are computed according to Equation A.4. Note that since $\mathbf{W}$ is a column vector, $\mathbf{h}^{(L-1)}\mathbf{W}$ is a scalar output and the only thing that is different for different probabilities are the biases $\mathbf{b}$. In the case of Cheng's method, $\mathbf{h}^{(L-1)}\mathbf{W}$ is really a vector of K cumulative probabilities. This means that it is a slightly more expressive model compared to POM (due to the increase in the number of learnable parameters) but at the cost of monotonic cumulative probabilities. However, for deep models – which usually have a very large number of learnable parameters – the benefit of $p(K-1)$ extra parameters would diminish as the network gets deeper. While the POM may seem tricky to implement due to its requirement that the biases be monotonic, it is actually quite simple to enforce without the need for constrained optimisation algorithms: suppose we have a vector $\mathbf{b}_0 \in \mathbb{R}^K$, we can define either $\mathbf{b} = \mathbf{b}_0^2\mathbf{U}$ or $\mathbf{b} = \mathbf{b}_0^2\mathbf{L}$, where $\mathbf{U} \in \{0,1\}^{K\times K}$ is an upper-triangular matrix of ones (if we want $\mathbf{b}$ to be monotonically increasing), and $\mathbf{L} \in \{0,1\}^{K\times K}$ is a lower-triangular matrix of ones (if we want monotonically decreasing biases). We end up squaring $\mathbf{b}$ so that we are not subtracting values when we multiply it with one of the triangular matrices.

Earlier, we asked the question as to why some of the techniques we presented end up defining cumulative probabilities which can then be reformulated as discrete probabilities for classification. One reason is actually through the latent variable intepretation of the proportional odds model (McCullagh, 1980). In the POM we assume that there is some latent variable $\mathbf{z} \in \mathbb{R}$ from which the discrete-valued class $y \in 0, 1, \ldots, K-1$ is derived. Concretely, we can think of there existing 'cut-off' points $\alpha_0, \ldots, \alpha_{K-2}$ where if z lies between α_i and α_{i+1} then the resulting label is $y = i$, i.e., $a_j \leq z \leq a_{j+1} \implies y = j$. Of course, in practice this is a bit of an idealisation, because often we do not know the latent variable z, and even if we did, it would make more sense to just model that instead since it would contain more

[4] The direction of the inequality is actually flipped here, but this is not a relevant detail.

information than a crude discretisation of z. This means that when we model $p(y \leq j|\mathbf{x})$ this is also equivalent to modeling $p(z \leq \alpha_j|\mathbf{x})$, where the α's thresholds that end up cutting the space of conditional probabilities. Note in our estimation of the POM we learn a bias vector $\mathbf{b}$ which is precisely an estimation of these α's! So from this, we can see that the idea of defining discrete probabilities from cumulative ones is a natural consequence of the latent interpretation of the POM.[5]

Lastly, another ordinal technique which is worth mentioning is the stick-breaking approach by Khan *et al.* (2012), which provides a reformulation of the multinomial logit (softmax) formulation in a way that is appropriate for ordinal problems. In the stick-breaking approach, we define discrete probabilities by defining a stick of unit length between 0 and 1, and sequentially breaking off parts of the stick which then become the discrete probabilities for that class. For example, suppose that $f(\mathbf{x}) \in \mathbb{R}^K$ was a vector denoting the output of our network. We can then define the stick length of the first class, i.e., its probability, to be $\sigma(f(\mathbf{x})_0)$, where $\sigma(\cdot)$ denotes the sigmoid nonlinearity. We can then define the second class probability as what was left over from that stick multiplied by the output of the second class, i.e., $(1 - \sigma(f(\mathbf{x})_0)\sigma(f(\mathbf{x})_1)$. For the third class probability we compute $(1 - \sigma(f(\mathbf{x})_0)(1 - \sigma(f(\mathbf{x})_1)\sigma(f(\mathbf{x})_2)$ and so forth, where the last class probability for $K - 1$ receives what is left over, i.e., $(1-\sigma(f(\mathbf{x})_0)...(1-\sigma(f(\mathbf{x})_{K-2})$. We can also re-write these probabilities as the following:

$$p(y = j|\mathbf{x}) = \frac{\exp(\sigma(f(\mathbf{x})_j))}{\prod_{i<j}(1 + \exp(\sigma(f(\mathbf{x})_i)))}, \tag{A.5}$$

where $\mathbf{h}(\mathbf{x})$ are the softmax inputs (logits). It can be shown that each output $f(\mathbf{x})_i$ is actually the log-ratio $f(\mathbf{x})_i = \log(\frac{p(y=k|\mathbf{x})}{p(y>k|\mathbf{x})})$, so these $f(\mathbf{x})_i$'s can be interpreted as defining decision boundaries that try to separate the k'th class from all the classes that come after it. Perhaps a nice property of this technique is that unlike POM, we obtain a slightly more expressive model since each class j gets its own scalar output $f(\mathbf{x})_j$. We also model the discrete probabilities directly instead of having to define cumulative ones first.

While we gave a somewhat comprehensive overview of ordinal techniques in regards to neural networks, none of these methods explicitly enforce unimodal probability distributions. Because of this, we proposed a completely different method through the parameterisation of the Poisson and binomial distributions rather than build on the works we have just described.

[5]This is also analogous to how we compute discrete probabilities for continuous distributions. If we want to compute the probability that z is in between i and j (where $j \geq i$), we simply perform the subtraction $p(z \leq j) - p(z \leq i)$.

Evaluation

In this work our primary evaluation metric is the quadratic weighted kappa, which is an appropriate metric to use for ordinal problems. This is because of it takes into account that some misclassifications are most costly than others, enforced through a quadratic penalty (hence the name). But it is also a useful metric because of class imbalance (which we have for the diabetic dataset); suppose we have a binary class dataset where 90% of cases full under the first class. A random chance classifier would simply predict the first class and obtain 90% accuracy. Unless one knows the class composition of the dataset, claiming a 90% accuracy can be misleading. Kappa however is what is called a 'chance-corrected' metric – that is to say, if we predict the majority class for all predictions, the resulting kappa would be zero.

The quadratic weighted kappa is defined as the following:

$$\kappa = 1 - \frac{\sum_{i,j} \mathbf{W}_{ij}\mathbf{O}_{ij}}{\sum_{i,j} \mathbf{W}_{ij}\mathbf{E}_{ij}} \tag{A.6}$$

It measures the level of agreement between two raters, $\mathcal{A}$ and $\mathcal{B}$ (where one rater is the ground truth and the other is the classifier). If $\kappa < 0$ then the classifier performs worse than random chance (according to the expected marginal distribution, which we explain shortly) and if $\kappa = 1$ then there is complete agreement between the two raters. Let us now explain what each term in this equation represents.

$\mathbf{O}$ is a $k \times k$ matrix where $\mathbf{O}_{ij}$ is a count of how many times an instance received a rating i by rater $\mathcal{A}$ and a rating j by rater $\mathcal{B}$. This is equivalent to $\{\tilde{p}(\mathcal{A} = i, \mathcal{B} = j)\}_{i,j=1}^{k}$, i.e., what is the (unnormalised) joint probability that rater $\mathcal{A}$ classifies an instance as class i and rater $\mathcal{B}$ classifies the same instance as class j. If we let $\mathbf{Y}$ be a $n \times k$ matrix where each row $\mathbf{Y}_i$ denotes the one-hot encoded label of a class, and $\mathbf{P} = \{p(\mathbf{Y}_i|\mathbf{X})\}_{i=1}^{n}$ be the corresponding matrix of predictions, then $\mathbf{O} = \mathbf{Y}^T\mathbf{P}$.

$\mathbf{E}$ is also a $k \times k$ matrix of 'expected' ratings, i.e., what if we assumed that $\tilde{p}(\mathcal{A} = i, \mathcal{B} = j) = \tilde{p}(\mathcal{A} = i) \times \tilde{p}(\mathcal{B} = j)$? Then in that case, we can simply construct the expected ratings matrix $\mathbf{E}$ (also a $k \times k$ matrix) by computing the outer product between the vector of column sums of $\mathbf{Y}$ and the vector of column sums of $\mathbf{P}$. Let us also normalise $\mathbf{E}$ so that it has the same total sum as $\mathbf{O}$, by dividing the matrix by the sum of all the elements in $\mathbf{O}$.

Lastly, let us define $\mathbf{W}$. This is simply a $k \times k$ matrix where $\mathbf{W}_{ij}$ denotes the cost associated with misclassifying class i as class j (and the converse). For a discrete classification, $\mathbf{W}_{ij} = 0$ for $i = j$ and 1 otherwise. For quadratic weighted kappa, $\mathbf{W}_{ij} = (i - j)^2$.

ANNEXE B Image-to-image translation

The following excerpt is from a recent paper we submitted to CVPR by Joel Moniz (lead author), myself, and Christopher Pal (research director). My contribution to this work was through the adversarial clean-up stages of the face frontalisation pipeline that Joel and Christopher developed.

not predicted, essentially dropping it.

As we operate on key-points, the actual warping of pixels can be performed with a high quality and high performance OpenGL pipeline that performs the warp separately from the rest of the architecture. This warping is also not needed during training. A key advantage of using this formulation is that underlying imagery of arbitrarily high resolution can be processed with no impact on the complexity of depth and geometry prediction operations.

The loss function of a Depth-Net is obtained by transforming the source face to match the target face using the simple squared error of the corresponding target object's key-point vector $\mathbf{x}_t = [x_t, y_t]^T$ and the source object's normalized key-point vector $[x_n, y_n]^T$, all normalized to the range [0, 1] by dividing with the total image size. The loss for one example where we predict depth and affine viewpoint geometry can therefore be expressed as:

$$\mathcal{L} = \sum_{i=1}^{K} \left|\left| \mathbf{x}_t^i - \mathbf{F}(\boldsymbol{I_s}, \boldsymbol{I_t}) \, [x_s^i \;\; y_s^i \;\; \mathbf{z}_\mathbf{p}^\mathbf{i}(\boldsymbol{I_s}, \boldsymbol{I_t})]^T \right|\right|^2 \quad (1)$$

2.2. Estimating viewpoint geometry as a second step

In this model variant, rather than using the predicted 3D affine transformation for pairs of objects, we use the predicted depths to estimate the affine geometry as a second estimation step. More precisely, given 3D points for a scene and the corresponding 2D points for a reference geometry it is possible to formulate the estimation of a 3D affine transformation as a linear least squares estimation problem. An overdetermined system of the form $\boldsymbol{A}\boldsymbol{m} = \boldsymbol{x}_g$ for this problem can be constructed as follows:

$$\begin{bmatrix} x_s^1 & y_s^1 & z_s^1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & x_s^1 & y_s^1 & z_s^1 & 0 & 1 \\ x_s^2 & y_s^2 & z_s^2 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & x_s^2 & y_s^2 & z_s^2 & 0 & 1 \\ & & & & \vdots & & & \\ x_s^K & y_s^K & z_s^K & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & x_s^K & y_s^K & z_s^K & 0 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x_t^1 \\ y_t^1 \\ x_t^2 \\ y_t^2 \\ \vdots \\ x_t^K \\ y_t^K \end{bmatrix}$$

This corresponds to an affine camera model followed by an orthographic projection to 2D. This setup also leads to the following closed form solution for the affine transformation parameters: $\boldsymbol{m} = [\boldsymbol{A}^\boldsymbol{T}\boldsymbol{A}]^{-\mathbf{1}}\boldsymbol{A}^\boldsymbol{T}\boldsymbol{x}_g$, where this pseudoinverse based transformation is parameterized by the reference points and their predicted depths.

2.3. Joint viewpoint and depth prediction

Our key observation is that one can alternatively use the closed form analytical solution for the least squares estimation problem as the underlying transformation of depth enhanced keypoints within the loss function. This leads to a special form of structured prediction problem for geometrically consistent depths and affine viewpoint geometry. For each image we therefore have a loss of the form: $\mathcal{L} =$

$$\sum_{i=1}^{K} \left|\left| \begin{bmatrix} x_t^i \\ y_t^i \end{bmatrix} - \begin{bmatrix} m_1 & m_2 & m_3 & t_x \\ m_1 & m_2 & m_3 & t_y \end{bmatrix} \begin{bmatrix} x_s^i \\ y_s^i \\ z_p^i(\boldsymbol{I_s}, \boldsymbol{I_t}) \\ 1 \end{bmatrix} \right|\right|^2$$

$$= \sum_{i=1}^{K} \left|\left| \boldsymbol{x}_t^i - \text{reshape}\big[[\boldsymbol{A}^\boldsymbol{T}\boldsymbol{A}]^{-\mathbf{1}}\boldsymbol{A}^\boldsymbol{T}\boldsymbol{x}_t\big]\boldsymbol{x}_s^i \right|\right|^2, \quad (2)$$

where the matrix $\boldsymbol{A}$ is parameterized as a function of $\boldsymbol{x}_s$ as shown in (2.2). In other words, ignoring the fact that the keypoints themselves are functions of the input images, and assuming looking at the Depth-Net inference functions purely in terms of keypoints, in contrast to the formulation above where the model predicts depth and the affine visual geometry parameters of $\mathbf{F} = \mathbf{F}(\{x_s^i, y_s^i, x_t^i, y_t^i\}_{i=1\ldots N})$ as a separate part of the Depth-Net, here we *parameterize* $\mathbf{A}$ as $\mathbf{A} = \mathbf{A}(\{x_s^i, y_s^i, \mathbf{z}_\mathbf{p}^\mathbf{i}(\{x_s^j, y_s^j, x_t^j, y_t^j\}_{j=1\ldots N}), x_t^i, y_t^i\}_{i=1\ldots N})$. We then reshape the least squares solution to form the affine matrix used to transform points within the loss. Since $z_s^i = \mathbf{z}_\mathbf{p}^\mathbf{i}(\{x_s^j, y_s^j, x_t^j, y_t^j\}_{j=1\ldots N})$ is predicted within the analytical formulation of the solution to the least squares minimization problem, we can backpropagate through the *solution* of a minimization problem that depends on the predicted depths.

Adversarial image-to-image transformation

Inevitably, some of the frontalized faces output from the Depth-Net geometry inference followed by textured mesh transformation look distorted. This happens due to the fact that when part of the face is obscured the appearance of surfaces of the face are unknown. To address this issue, we take a CycleGAN [28] approach. That is, we use an adversarial image-to-image transformation network which serves to repair the appearance of faces that have undergone frontalization through an inferred 3D model and a 3D warp of a textured triangular mesh. Importantly, the CycleGAN approach is an adversarial method which allows one to perform image transformation between two domains without the requirement of paired data. In our case, one domain is simply photos of frontal faces, while the other is faces that have been frontalized. This naturally defines the sets of good and bad quality frontal face images and we can leverage both sets of data to learn a GAN to 'clean up' the frontalized faces.

Suppose we have some images belonging to one of two sets $\mathbf{x} \in X$ and $\mathbf{y} \in Y$, where $\mathbf{x}$ denotes a poor quality face and $\mathbf{y}$ a high quality one. We wish to learn two functions $F : X \to Y$ and $G : Y \to X$ which are able to map an image from one set to the corresponding image in the other.

Correspondingly, we have two discriminators D_X and D_Y which try to detect whether the image in that particular set is real or generated. Note that while we are only interested in the function $F : X \rightarrow Y$ (since this maps from 'bad' face to 'good' face) the formulation of CycleGAN requires that we learn mappings in both directions during training in order to prevent F from learning trivial mappings (i.e. mode dropping). We optimize the following objectives for the two generators F and G:

$$\min_{G,F} \mathbb{E}_{\mathbf{x},\mathbf{y}} \Big[\ell(D_X(G(\mathbf{y})), 1) + \ell(D_Y(G(\mathbf{x})), 1) + \lambda ||\mathbf{y} - F(G(\mathbf{y}))||_1 + \lambda ||\mathbf{x} - G(F(\mathbf{x}))||_1 \Big] \quad (3)$$

And the following for the two discriminators D_X and D_Y:

$$\min_{D_X,D_Y} \mathbb{E}_{\mathbf{x},\mathbf{y}} \Big[\ell(D_X(\mathbf{x}), 1) + \ell(D_X(G(\mathbf{y})), 0) + \ell(D_Y(\mathbf{y}), 1) + \ell(D_Y(F(\mathbf{x})), 0) \Big], \quad (4)$$

where 0/1 denote fake/real, $\ell()$ is the squared error loss (since we use LSGAN [18]), and λ is a coefficient for the cycle-consistency (reconstruction) loss. Once the network has been trained, we can disregard all other functions and use F to clean up faces which are low quality due to artifacts from warping.

3. Related Work vs. Our Approach

3.1. 3D transformation and faces

While there is a large literature on 3D facial analysis, many standard and well known techniques are not applicable to our setting here – consisting of estimating depths from a single input image. As an example, Morphable models [1] cover a wide variety of approaches which are capable of high quality 3D reconstructions, but such methods usually require 3D face scans or reconstructions from multiview stereo to be assembled so as to learn complex parametric distributions over face shapes. Our single image setting also eliminates many other standard computer vision techniques from being applicable, such as structure from motion methods.

One of the closest approaches to our own of which we are aware is that of [6] on viewing real world faces in 3D. Like our work this approach does not require aligned 3D face scans, highly engineered models or manual interventions. In his work they make the observation that if 2D keypoints can be obtained from a single input image of a face then if these keypoints are matched to an arbitrary 3D reference geometry, then standard camera calibration techniques can be used to estimate plausible intrinsics and extrinsics of the camera. This allows the estimated camera matrix, 3D rotation matrix and 3D translation vector to be used to transform the reference 3D model to the pose of the query image from which an approximate depth can be obtained. [6] then bridges the difference between the depth estimate obtained from the spatially transformed reference model and the query face using a coordinate descent based joint depth and appearance optimization. Some more recent work has explored the use of a single unmodified 3D surface as an approximation to the shape of all input faces [7]. In contrast, in our approach we infer an image specific 3D model as well as geometric transforms allowing for the retargeting of the inferred model for both viewpoint or even another target facial geometry. The work in [7] also underscored the issue of visibility when faces are not frontal facing. In our work this issue manifests as incorrect appearance information obtained from pixels used for the initial texturing of a 3D mesh. We address the issue in our work here with adversarial repair.

In the high profile DeepFace work of [22], the authors focused extensively on face frontalization to improve the performance of a convolutional neural network based face verification system. They first detect, crop and align a face inplane. They use a 3D mask composed of facial key points, detect the corresponding locations of these key points in the image, and map the 3D model onto the plane of the face. They then use this model to do a piecewise-affine transform on the face to frontalize it. They follow an end-to-end architecture, except that they take advantage of the fact that the images are now frontalized to add a few "locally connected" layers in between, which are convolutional feature maps, but in which each location in each feature map learns a different set of weights. In contrast, in our work we do not require a 3D reference face model, we only require an arbitrary 2D reference keypoint geometry - no reference appearance or textured 3D model is needed. We also simply use an affine camera model to account for potentially dramatic differences in query versus reference geometry. Our approach is therefore less tied to faces in particular and is applicable to any sort of object where 2D keypoint locations can be placed into semantic correspondence and geometric warping under a 3D affine model is reasonable.

Spatial Transformer Networks (STNs) [11] work either directly on input data, or on intermediate representations, to enhance the spatial invariance of CNNs, and to improve invariance to rotations, scaling and warping. They do this by learning and applying an input-dependent backpropable affine transformation either directly on the input immage, or on an intermediate representation of the image (generated after the input has been passed through a part of a CNN pipeline). While STNs represent an excellent step towards automating pre-processing tasks such as cropping and rotation, this model fails for a crucial aspect of the normalization: out-of-plane rotations.

scaled down to the size 80×80 and converted to greyscale, following which they are passed through the RCN to obtain $N = 68$ key points on each image. The RCN is trained exactly as described in [9], using the 300W dataset [20].

The key-point only variant of our model involves concatenating all detected key-points and passing them through a two-layer deep fully connected network (with 256 and o hidden units respectively, where o depends on whether we are predicting only the depth, when $o = N$, or the depth and an affine transform as well, when $o = N + 8$).

As discussed above, it is possible to augment these models with a Siamese CNN module. For our exploration here model variants that also use the image pass both the source and the reference images through three conv-maxpool ((32, 4, 2), (48, 3, 2), (64, 2, 2) respectively for the representation (num_filters, filter_size, pool_size), all filters being square) layers with shared weights. The output of this is concatenated (much like in a Siamese architecture, as in [2], [3]) before passing them into a 4-layered fully connected network (with respective output sizes 2048, 512, 256 and o). The key-points are injected between the second and third layers (by concatenating them to the output of the second layer). We explore these Siamese CNN augmented model variants in Model 3 and 5 in Table 1.

We set the initial learning rate of 0.001 and use a nesterov momentum optimizer (with a momentum of 0.9) in all our experiments. With the exception of the last layer, we initialize all weights using a Glorot initialization scheme ([4]) (with the weights sampled from a uniform distribution) with a ReLU gain ([8]), and all biases to 0, and apply a ReLU non-linearity after every layer. In the final output layer, we do not apply any non-linearity and initialize the weights to 0. The biases of units that represent depths are initialized to a random Normal distribution with $\mu = 0$ and $\sigma = 0.5$, while those that form the predicted affine transform are initialized with the equivalent of a "flattened" identity transform. All models have been trained for 500 epochs.

We point out that except for a comparison between learning rates in the set $0.01, 0.001, 0.0001$ over a few (less than 10) epochs to find a learning rate that the model seems to train well with, we have not performed a hyperparameter search, and anticipate that the performance of the model can be made *even* better by searching the hyperparameter space on a per model basis and by using deeper (or modified) architectures.

Unless otherwise specified, for the experiments described below, we use a subset of the VGG dataset [19] architecture, training and validating on all possible pairs of images belonging to the same identity for 2401 identities, with roughly 20 images per identity, yielding 322227 train and 43940 validation pairs).

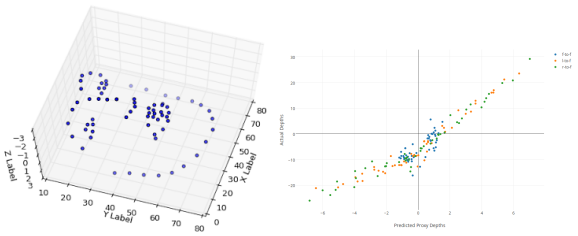


Figure 3. (left) Depth inferred for a randomly sampled face (right) A comparison of inferred depths to ground truth for two faces. Predictions are on the x-axis, the ground truth is on the y-axis.

4.2. Visualizing Depth Predictions

In Figure 3 (left) we show the depths that have been inferred for a randomly selected face. In Figure 3 (right) we compare the predicted depths from two randomly selected faces from the 3DFAW dataset ([12], [27], [25], [5]) with ground truth depths (which are provided with this dataset). We see that predictions on the x-axis match reasonably well with the ground truth on the y-axis, up to a scale factor. This unidentifiable scale factor is to be expected as we discussed above. Note that for this task, we retrain the model on the VGG dataset, but by using a version of our model that predicts 66 proxy depth values instead of 68. In this case, we generate the key-points as before, using an RCN trained on the iBug dataset, but we discard two key-points. It is useful to note that for extreme pose differences, face contour points do not correspond exactly, so some errors arise from this fact, and we show the graph without these points.

In Figure 2 (left) we compare histograms of the mean squared error for the models given in Table 1. As described at the start of the Experiments section, we train and test various versions of our model on a sub-set of the identities of the VGG dataset (with around 20 images present per identity), and measure the registration error (in the form of both the MSE and the Euclidean distance normalized by the intra-ocular distance, similarly to [9], for all possible pairs of faces belonging to the same identities in the respective sets. Thus, perfect registration should yield an error close to 0 – but not exactly zero since facial expressions may be different in each image. We see a clear progression as our models capture the interaction between depth estimation and visual geometry estimation and perform the estimation more directly. In Figure 2 (right) we provide some examples of a source face (top left) and a target face (bottom left), their pre-registration alignment (top right) and the transformation of the source to the target (bottom right).

4.3. Adversarial Appearance Repair

We use a simple heuristic based on keypoint distances to extract frontal faces from the VGG dataset. From this process we extracted 622 unmodified frontal faces to serve as the set of 'good' frontal faces. We used 1,480 frontalized faces as the set of'bad' faces. This set thus contains

Figure 4. A visualization of frontalized 3D faces using a Depth-Net followed adversarial appearance repair using a CycleGAN trained on the CelebA faces. For all but the last row however, the input faces are from the VGG dataset. The last row uses CelebA test set faces.

Figure 5. Left to right: source face; target face; warp; warp with the source highlighted; warped result pasted on target.

a mixture of relatively high quality warps with minor artifacts as well as significant distortions due to the 3D warping procedure. While the trained CycleGAN model performed decently in fixing the warps, we instead decided to employ the same keypoint detection and frontalization pipeline on the CelebA dataset so that we would have a much larger set of good/bad frontal faces to work with. In doing so, we ended up with a significantly larger dataset, with 43,114 good frontal faces and 8,310 bad ones.

The generators we trained for CycleGAN are FCN-like networks [17] employing both short and long skip connections, while the discriminators are 'patch GANs' just like in [28], where the network simply consists of several conv-BN-relu blocks and the output of the network is a set of patches. We employ instance normalization [23] and leaky ReLUs for all networks, use $\lambda = 10$ for the reconstruction loss, and train using Adam [16] with learning rate $\alpha = 2 \times 10^{-4}$, $\beta_1 = 0.5$ with batch size 32. (For additional details regarding the architecture we refer the reader to the supplementary material for precise definitions of the networks employed.) The results from this can be seen in Figure 4. Though we trained the network only on CelebA, all rows but the last were evaluated on faces from the VGG dataset.

4.4. Face Replacement

Here, we take a source and target image and extract their keypoints using a Recombinator Network. We pass the source and target through our Depth-Net to obtain the depth for each of the keypoints in the source image and a global 3D affine transform to map the source to the target face. Pixels from the original source image are used to form the texture of a triangulated 2D mesh based on the keypoints. The predicted depths are assigned to the keypoints, with the depth of the points in between being interpolated, to form a 3D mesh, which is warped in 3D and orthographically projected to the target geometry using the predicted affine model for visual geometry. Some examples of this procedure are shown in Figure 5.

5. Conclusion

We have proposed a novel neural network approach to 3D face model creation which enables relative pose normalization or frontalization without the use any ground truth depth data. We achieve our best quantitative keypoint registration results using our novel formulation for predicting depth and 3D visual geometry simultaneously, learned through backpropagating through the analytic solution for the visual geometry estimation problem expressed as a function of predicted depths. Further, we also see a dramatic increase in the visual quality of frontalized faces afforded by our adversarial appearance repair approach.